Planning Optimal Motions for Anthropomorphic Systems

Antonio El Khoury

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par:

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par:
Antonio EL KHOURY
le lundi 3 juin 2013

Titre:

Planification de Mouvements Optimaux pour des Systèmes Anthropomorphes
Planning Optimal Motions for Anthropomorphic Systems

École doctorale et discipline ou spécialité:
EDSYS: Robotique et Informatique

Unité de recherche:
LAAS-CNRS

Directeurs de Thèse:
M. Florent LAMIRAUX
M. Michel TAIX

Autre membres du Jury:

M. Patrick DANES	Président
Mme Maren BENNEWITZ	Rapporteur
M. Abderrahmane KHEDDAR	Rapporteur
Mme Brigitte D'ANDREA-NOVEL	Examinateur
M. Timothy BRETL	Examinateur
M. Rodolphe GELIN	Examinateur

Résumé

L'objet de cette thèse est le développement et l'étude d'algorithmes de planification de mouvements optimaux pour des systèmes anthropomorphes sous-actionnés et hautement dimensionnés, à l'instar des robots humanoïdes et des acteurs virtuels. Des méthodes de planification aléatoires et de commande optimale sont proposées et discutées. Une première contribution concerne l'utilisation d'une méthode efficace de recherche dans un graphe pour l'optimisation de trajectoires de marche planifiées pour un système modélisé par sa boîte englobante. La deuxième contribution concerne l'utilisation de méthodes de planification aléatoires sous contraintes afin de planifier de façon générique des mouvements corps-complet de marche et manipulation. Enfin nous développons une approche algorithmique qui combine des méthodes de planification aléatoires sous contraintes et de commande optimale. Cette approche permet de générer des mouvements dynamiques, rapides, et sans collision, en présence d'obstacles dans l'environnement du système.

Abstract

This thesis deals with the development and study of algorithms for planning optimal motions for anthropomorphic systems, which are underactuated and highly redundant systems, such as humanoid robots and digital actors. Randomized motion planners and optimal control methods are proposed and discussed. A first contribution concerns the use of an efficient graph search algorithm in order to optimize walk trajectories that were previously obtained for a bounding-box representation of the system using randomized motion planners. The second contribution develops the use of constrained randomized motion planners in order to plan in a generic way whole-body motions that involve both walking and manipulation. Finally we develop an algorithmic approach which combines constrained randomized motion planners and optimal control methods; this approach allows the generation of dynamic, fast and collision-free motions for anthropomorphic systems in the presence of obstacles.

Remerciements

J'ai profité d'un accueil exceptionnel au LAAS-CNRS durant mes trois années de thèse. C'est pourquoi je tiens tout d'abord à remercier ses directeurs successifs Raja Chatila, Jean-Louis Sanchez et Jean Arlat, le directeur du thème Robotique Rachid Alami, et plus spécifiquement les directeurs successifs du groupe Gepetto Jean-Paul Laumond et Philippe Souères.

Je tiens à exprimer toute ma gratitude envers mes directeurs de thèse Florent Lamiraux et Michel Taïx. Leur confiance, leurs connaissances approfondies, leurs précieux conseils et leur sympathie ont directement contribué au travaux que j'ai effectués et au plaisir que j'en ai tiré.

C'est un honneur d'avoir Maren Bennewitz et Abderrahmane Kheddar comme rapporteurs de ma thèse, et je les remercie sincèrement pour leur relecture attentive de mon manuscrit. Je remercie également Brigitte d'Andréa-Novel, Timothy Bretl, Patrick Danès et Rodolphe Gelin d'avoir accepté de faire partie de mon jury de thèse, ainsi que pour leurs remarques et discussions intéressantes.

J'ai eu la chance d'effectuer un séjour scientifique à l'Université de Heidelberg dans le groupe ORB . Je tiens à remercier Katja Mombaur pour ses précieux conseils et l'excellent accueil qu'elle m'a réservée.

J'ai été très marqué par l'esprit d'équipe qui règne dans le groupe Gepetto, et espère y avoir contribué durant ces trois années. Je remercie chaleureusement Nicolas Mansard et Olivier Stasse qui, sans être directement impliqués dans mes travaux, m'ont fourni tout le soutien scientifique et technique dont un doctorant pourrait rêver.

Ces trois dernières années sont passées rapidement; ceci est principalement dû à mon côtoiement au quotidien de doctorants et stagiaires sympathiques et brillants, dont la bonne humeur a ajouté encore plus de plaisir à ces travaux de recherche. J'ai eu la joie de collaborer étroitement avec Sébastien Dalibard, Martin Felis, David Flavigné et Thomas Moulard; je les remercie pour leur dévouement à nos travaux ainsi que pour leur amitié.

Je me sens privilégié d'avoir pu rencontrer de glorieux anciens, qui m'ont inculqué les préceptes de l'esprit d'équipe et de la bonne ambiance. Merci donc à Duong, François, Layale, Manish, Maxime, Nicolas, Oussama, Samory, Sovan, Valentin, Wassim, et Wassima. Je remercie également tous les actuels membres qui

perpétuent la tradition: Aiva, Andreas, Arturo, Francesco, He, Henning, Léo, Laurent, Mauricio, Mehdi, Olivier, Oscar, Perle, avec une mention spéciale pour Justin Carpentier, Olivier Roussel, et Jorrit T'Hooft qui ont consacré une semaine de leur temps à la construction d'un magnifique mur de brique. Je leur souhaite à tous bonne continuation.

S'il arrive un jour à lire et comprendre ce manuscrit, je tiens à remercier le robot HRP-2 14 pour avoir supporté, sans jamais se plaindre, les collisions, les chutes, et tous les mouvements "inhumains" que je lui ai fait faire.

Je souhaite exprimer toute ma reconnaissance à ma famille, et plus particulièrement à mes parents et mon frère, pour leur amour, leurs conseils attentionnés et leur soutien constant.

Enfin je souhaite remercier Maya pour son soutien, son amour, et pour m'avoir accompagné, malgré la distance, durant ces trois dernières années qui ont mené jusqu'à ma soutenance. Et la fin n'est que le début.

```
(0,0)
/)_)
 ""
```

Contents

List of Figures

List of Tables

List of Algorithms

Notation table

This is a brief review of notations used in this thesis, listed by appearance order.

Term	**Meaning**
$\mathbf{q}$	Configuration, generalized position
$\dot{\mathbf{q}}$	Generalized velocity
$\ddot{\mathbf{q}}$	Generalized acceleration
$\dddot{\mathbf{q}}$	Generalized jerk
$\boldsymbol{\tau}$	Generalized torque
$\mathcal{CS}$	Configuration space
$\mathcal{SS}$	State space
$\mathcal{CS}_{free}$	Free configuration space
RRT	Rapidly-exploring Random Trees
PRM	Probabilistic Roadmaps
DoF	Degrees of Freedom
B_i	Body i of robot
J_i	Joint i of robot
CoM	Center of Mass
ZMP	Zero-Moment Point
g	Gravity constant
RO	Random Optimization
RSO	Regular Sampling Optimization
IK	Inverse Kinematics
$\mathcal{M}$	Manifold
QP	Quadratic Programming
NLP	Nonlinear Programming
SQP	Sequential Quadratic Programming
IPM	Interior-Point Method
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
OCP	Optimal Control Problem
RNEA	Recursive Newton-Euler Algorithm
CRBA	Composite Rigid Body Algorithm
ABA	Articulated Body Algorithm

Introduction

While the term *robotics* was first coined by Isaac Asimov some seventy years ago, and while the first industrial robots were developed in 1961, the description of automaton mechanisms can be dated back to the tenth century B.C., which is a proof of Man's long-time fascination with robots. The first robots had limited capabilities, mainly focused on moving some of their limbs in order to *act* on their environment or simply to entertain. Starting from the middle of the twentieth century, technological advances in electronics allowed the creation of sensors which allowed robots to *perceive* their surroundings (including their own body), i.e. to build a useful representation of them. Finally, thanks to the breakthroughs in the Computer Science and Mathematics fields, the means to *decide* how to act based on perception were given to robots (and roboticists too!). This decision process is known as *planning*.

Anthropomorphic Systems in Robotics

Anthropomorphic systems can be described as systems which are made to look like humans with respect to their mechanical structure and their abilities. In robotics, they usually have similar perception systems, such as visual sensors in the head, tactile and inertial sensors. Also, they usually have two arms and more importantly two legs, which sets them apart from wheeled robots. These features are such that it is very complex to implement a perception-planning-action on anthropomorphic systems (or humanoid robots).

So why do we bother designing and studying anthropomorphic systems? Simply because it is fun! As the reader may still be skeptical with respect to this argument, there are luckily many applications for humanoids robotics. First of all, as humanoid robots are made to look like humans, they have similar abilities that can allow them to walk, run, jump, climb, write, carve, manipulate, etc. There are of course highly specialized robots that can achieve each one of those tasks, but very few exhibit such a high versatility. Also, we are witnessing a shift from industrial robotics to personal robotics. Humanoid robots can have access to the same environments as humans, and they are made to look like them and to be easily accepted: they can therefore be used as companions, assistants, guides, teachers, etc. in various situations.

From a research point of view, anthropomorphic systems are very interesting as they give researchers the means to devise new algorithms to control them and

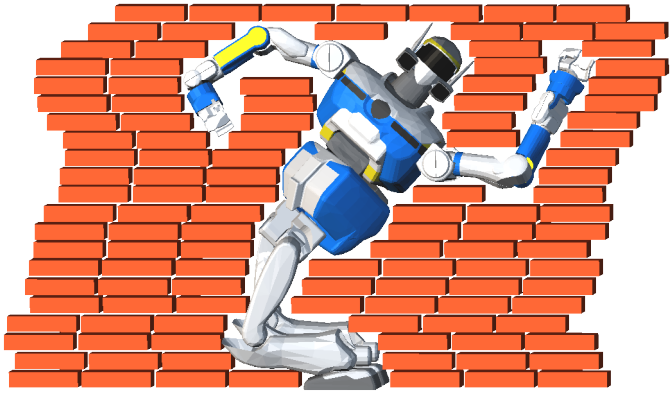


Figure 1: The human-like mechanical structure of humanoid robots enables them to accomplish complex tasks, such as avoiding obstacles in constrained environments.

make them walk and manipulate objects without loosing their balance, allowing the generation of human-like motions. Conversely, humanoid robots are perfect tools for neuroscientists to understand human motion and perception, as well as the invariants behind them; for instance, moving certain actuators while keeping others at a fixed position is very easy in robotics, and this allows decoupling different behaviors in order to observe them more clearly. Humanoid robots also tend to be more cooperative (so far) than humans or monkeys when it comes to accomplishing tedious and repetitive tasks during experiments.

In this thesis, we focus on developing new algorithmic tools for *planning optimal motions for anthropomorphic systems*, as well as applying them on humanoid robots.

Problem Statement

The problem of motion planning for anthropomorphic systems can be defined as the following: given a starting configuration, say a position and posture, we would like an anthropomorphic system, say a humanoid robot or a digital actor, to reach a goal configuration if it is possible. Obviously such a system cannot move instantaneously, so it will have to travel continuously in the environment surrounding it to reach its goal. The environment will usually not be empty, as it will contain at least the ground that supports the system. It might also contain moving or static entities which we do not want the system to collide with in order to avoid damaging either the entities, the system, or both. Additionally, we want to avoid self-collisions between the different limbs of the system. Finally, as anthropomorphic systems rely on making and breaking contact with their environment in order to move, great care must be given to make sure the system can realize the required motions and does not fall. This requires a good definition of a system *balance* and the means to ensure it. Therefore, finding a solution to the problem of humanoid walk planning

consists in finding a continuous motion connecting the start configuration to the goal configuration, such that the anthropomorphic system is never in collision with the environment or itself when executing this motion, and such that it is always balanced.

While the found solution is guaranteed to be collision-free, we still know nothing about its "quality". In the case of anthropomorphic system motion, the notion of quality can be linked to how close a motion is to real human motion, i.e. one which a human being would have made if he was put in the same conditions. Obtaining high-quality motions is desirable since humanoid robots are bound to move in man-made environments such as homes, offices, and factories and because it could help them blend in among humans more seamlessly. Subsequently, we would like to impose additional constraints on the motion planning problem in order to find motions that are both collision-free and optimal with respect to a certain cost measure. We refer to this problem by the name of optimal motion planning.

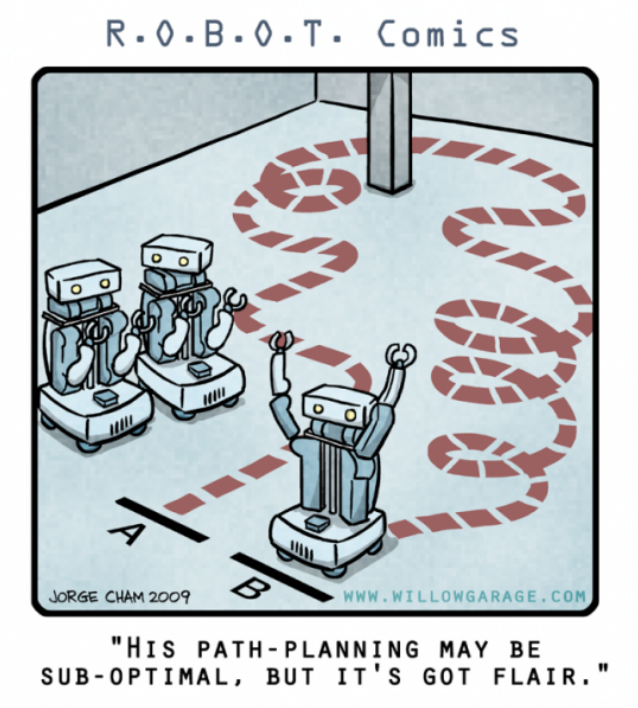


Figure 2: A robot solves a motion planning problem. The solution is collision-free, but obviously the robot can do much better (Jorge Cham, PhD Comics).

Contributions

The first contribution of this thesis is a heuristic and efficient optimization method that takes as input a path computed for the robot bounding box, and produces a

path where a discrete set of configurations is reoriented using an A* search algorithm. The resulting trajectory leads to a significantly shorter walking time. This method is validated on various scenarios.

The second contribution is a whole-body motion planner for humanoid robots which computes collision-free walking trajectories, based on exact models of both the robot and its environment. It is used to solve manipulation tasks that may require walking. The first stage of our algorithm uses a sampling-based constrained motion planner and computes a collision-free statically balanced path for a robot which can be fixed or sliding on the ground. The formal proof that dynamic walking makes humanoid robots small-space controllable is then established; this directly implies that this first path can always be approximated by a dynamically balanced, collision-free walking trajectory. This well-grounded method is implemented and the results are validated on several environments.

A new framework for optimal motion planning is proposed as a *third contribution*. Given a humanoid robot geometric and dynamic model, an exact model of the environment, start and end configurations, and a robot contact stance, we first plan a collision-free statically balanced path that satisfies all kinematic constraints. We convert the path to an initial trajectory using a suitable time parametrization, and we then optimize it to generate a locally-optimal collision-free dynamically-balanced trajectory. This involves both finding a new time parametrization for the trajectory, and reshaping the path in a geometrical sense; thus, it is not simply a problem of optimal path tracking. In order to ensure (self-)collision avoidance during the optimization process, we choose to model distance constraints using bounding capsules around the robot exact body geometries. We provide an automatic bounding capsule generation tool; it relies on a numerical optimization problem formulation that allows us to find the minimum-volume capsules around bodies which are modeled by polyhedra. The capsules allow us then to enforce collision-avoidance constraints between the robot, the obstacles and itself.

Outline of This Thesis

The thesis is organized by contribution order, and the related work is explained when needed. Chapter 1 describes the path optimization method for humanoid walk planning. Chapter 2 deals with the second contribution, namely whole body motion planning and the associated small-space controllability proof. Finally, the optimal motion planning framework and the automatic bounding capsule generator are detailed in Chapter 3.

Publications in This Thesis

Journal

– Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Alireza Nakhaei, Michel Taïx and Jean-Paul Laumond. **Dynamic Walking and Whole-Body Motion Planning for Humanoid Robots: an Integrated Approach,** *International Journal of Robotics Research, 2013.*

Peer-Reviewed International Conferences

– Antonio El Khoury, Michel Taïx and Florent Lamiraux. **Path Optimization for Humanoid Walk Planning: an Efficient Approach,** *International Conference on Informatics in Control, Automation and Robotics, 2011.*

– Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Michel Taïx and Jean-Paul Laumond. **Small-Space Controllability of a Walking Humanoid Robot,** *IEEE International Conference on Humanoid Robots, 2011.*

– Antonio El Khoury, Florent Lamiraux and Michel Taïx. **Optimal Motion Planning for Humanoid Robots,** *IEEE International Conference on Robotics and Automation, 2013.*

Videos of Simulations and Experiments

Videos of simulations and experiments realized in this thesis can be found at this address:
`http://www.laas.fr/~aelkhour/video/thesis.html`

Software Contributions in This Thesis

Several open-source software packages were created or extended throughout this thesis. Some of them include:

– *kws-hash-optimizer*: a package for an efficient path optimization method, which is described in Chapter 1.

– *hpp-wholebody-step-planner*: a package for humanoid whole-body motion planning. It implements the algorithm that is presented in Chapter 2 of this thesis.

– *roboptim-capsule*: an automated bounding capsule generator over polyhedrons. It is based on an optimization formulation, which is presented in Chapter 3.

– *MetaPOD* (Meta-Programming Optimized Dynamics): a template-based C++ implementation of [Featherstone 08]. It is used in the optimal control formulation which is described in Chapter 3.

Chapter 1

Path Optimization for Humanoid Walk Planning: an Efficient Approach

This chapter deals with path optimization for humanoid walk planning in cluttered environments. Under the assumption that the humanoid robot will walk on a flat floor in a perfectly modeled static environment, it presents a heuristic and efficient optimization method that takes as input a path computed for the humanoid bounding box, and produces a path where a discrete set of configurations is reoriented using an A* search algorithm. A pattern generator is then used to generate a trajectory that minimizes walking time. This method is validated in various scenarios on the humanoid robot HRP-2.

1.1 Motion Planning in the Configuration Space

The problem of motion planning is now well formalized in robotics and several books present the various approaches [Latombe 91, Choset 05, LaValle 06]. One particularly useful concept is the one of *configuration space* $\mathcal{CS}$ [Lozano-Perez 83], which is the set of all configurations $\mathbf{q}$ of a robot R; $\mathbf{q}$ is a vector comprised of the n independent *degrees of freedom (DoF)* that are sufficient to uniquely identify the full state of the robot at each instant. $\mathcal{CS}$ defines then a manifold of dimension n. Some of the robot body positions can generate (self-)collisions; the equivalent configurations will be said to be in collision, and the set of all configurations in collision is denoted by $\mathcal{CS}_{obs} \subset \mathcal{CS}$. Its complement is denoted by $\mathcal{CS}_{free}$ and is called the *free configuration space*. Using these notations, we can redefine the motion planning problem as the answer to the following question: is there a continuous path $P : [0, 1] \rightarrow \mathcal{CS}_{free}$ that connects a start configuration $\mathbf{q}_s$ to a goal configuration $\mathbf{q}_g$, and what is it?

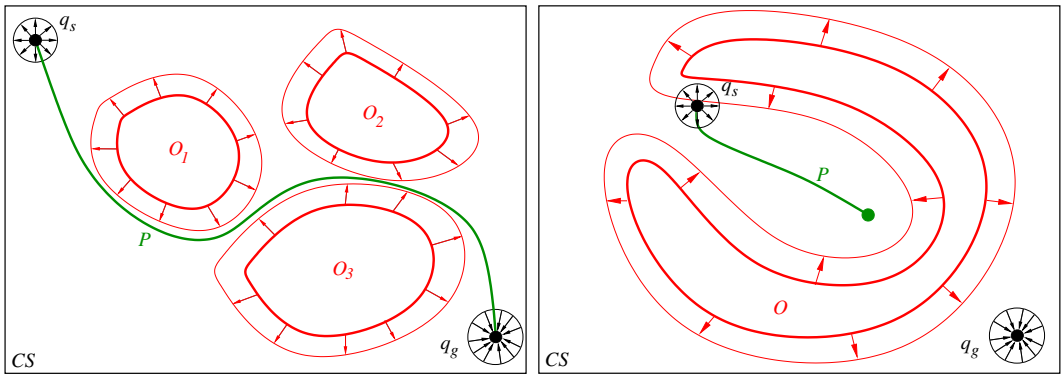


Figure 1.1: Left: A valid path is computed by the deterministic algorithm. Arrows show the attractive and repulsive potential fields. Thin lines show the potential lines. Right: Example of problem where the stable local minimizer does not coincide with the goal configuration, which is the global minimizer. P is thus not a solution to the path planning problem.

1.1.1 Deterministic Algorithms

This question can be answered through the use of deterministic algorithms; for a given number of tries, they will always compute the same valid path P.

One class of algorithms, mainly developed in the past 30 years, relies on representing $\mathcal{CS}_{obs}$ explicitly in order to build a graph, also called roadmap, that represents the connectivity of $\mathcal{CS}_{free}$. Solving the motion planning problem then boils down to a graph exploration to connect $\mathbf{q}_s$ to $\mathbf{q}_g$. A non-exhaustive list of these methods includes cellular decomposition, Voronoi diagrams, visibility graphs, and Canny's algorithm [Goodman 04].

Such algorithms offer the nice property of completeness, i.e. they can always provide an answer to the motion planning problem as defined previously. But while they work well for solving path planning problems in low-dimensional configuration spaces, using them in high-dimensional configuration spaces is either impossible by design, or is computationally expensive, as building $\mathcal{CS}_{free}$ requires finding its frontiers, and computation time is at best exponential with respect to the dimension of $\mathcal{CS}$.

Other approaches are inspired from real-time motion generation techniques, such as the one detailed in [Khatib 85], which consists in assigning artificial attractive potentials on the goals, and repulsive ones around the start configuration and the obstacles. The robot is then subject to forces that will direct it from the start configuration towards the goal configuration. However, because of the locality of the planner, a path may be computed while not being a solution to the path planning problem. This can happen in a maze-like environment when a stable equilibrium point other than the goal configuration is found (see Figure 1.1).

1.1.2 Sampling-based Algorithms

Deterministic algorithms rapidly reach their limit when the configuration space dimension rises above 4. Computation speed plays a big part in choosing which algorithm to use for path planning problems, as many applications require, or at least aim for, real-time resolution. In this perspective, sampling-based algorithms, such as Probabilistic Roadmaps (PRM) [Kavraki 96] or Rapidly-exploring Random Trees (RRT) [Kuffner 00], were developed in the past fifteen years.

Instead of trying to build an explicit representation of $\mathcal{CS}_{free}$, sampling-based algorithms rely on approximating the connectivity of $\mathcal{CS}_{free}$ through rejection sampling: random configurations $\mathbf{q}_{rand}$ are sampled in $\mathcal{CS}$, and efficient Boolean collision detection techniques [Hudson 97, Gottschalk 96] reject configurations that produce collisions, keeping only configurations $\mathbf{q} \in \mathcal{CS}_{free}$.

The classic RRT algorithm, as presented in [Kuffner 00], make use of the Voronoi bias to efficiently explore $\mathcal{CS}_{free}$ and grow a random tree in it. Each iteration of the algorithm attempts to extend the tree by adding new vertices in the direction of a randomly selected configuration $\mathbf{q}_{rand}$. Algorithm 1 shows the pseudo-code of the RRT algorithm. It takes as input an initial configuration $\mathbf{q}_s$ and grows a tree $\mathcal{T}$ rooted in $\mathbf{q}_s$.

Algorithm 1 RRT($\mathbf{q}_s$)

$\mathcal{T}$.Init($\mathbf{q}_s$)
for $i = 1$ to K **do**
  $\mathbf{q}_{rand} \leftarrow$ Rand($\mathcal{CS}$)
  $\mathbf{q}_{near} \leftarrow$ Nearest($\mathbf{q}_{rand}$,$\mathcal{T}$)
  Extend($\mathcal{T}$,$\mathbf{q}_{near}$,$\mathbf{q}_{rand}$)
end for

One way to make the RRT algorithm more efficient is to grow trees from both the initial and goal configurations, see Figure 1.2. This was first proposed in [Kuffner 00].

While not being complete (i.e. they cannot tell whether a solution exists or not), sampling-based algorithms have the weaker property of probabilistic completeness: if a solution path exists, then the algorithm will be able to compute it with a probability of 1 when the number of iterations K reaches infinity. In practice, these algorithms can compute paths in complex real-life environments in a reasonable time on regular computers and have been used to solve problems for various systems, ranging from 6-DoF floating objects, to 50-DoF anthropomorphic systems, to 1000-DoF proteins.

1.1.3 Path Optimization

As shown in Figure 1.2, RRT returns the shortest path P inside the graph that connects $\mathbf{q}_s$ to $\mathbf{q}_g$. Due to the probabilistic nature of RRT, it is clear that P is not

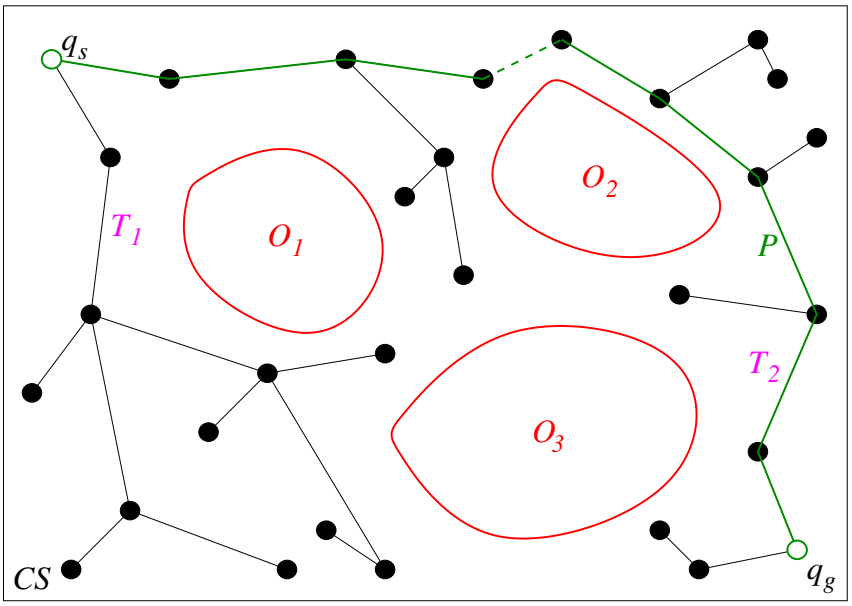


Figure 1.2: A valid path (in green) computed with a bidirectional RRT planner. q_{start} and q_{goal} are the roots of the trees $\mathcal{T}_1$and $\mathcal{T}_2$ respectively. The algorithm keeps diffusing both trees until they can be connected together with an edge (in dashed green).

optimal in terms of length. Path optimization methods take a valid, i.e. collision-free, path as input and try to shorten it while making sure that the output path is still valid.

Greedy Optimization

A greedy optimizer, such as the one shown in Figure 1.3, left, uses the greedy approach to shorten and smooth a path. First, it tries to connect directly $\mathbf{q}_s$ to $\mathbf{q}_g$; if the path is not collision-free, it tries to connect $\mathbf{q}_s$ to the node preceding $\mathbf{q}_g$, and so on until it reaches $\mathbf{q}_s$. This process is then restarted similarly on the following nodes.

Random Optimization

In the case of the greedy optimizer, the nodes that are in the optimized path are also nodes of the input path. Random Optimization (RO) tries to bypass some nodes and keeps the rest. While this simple method runs very fast, it does not always give the best possible path. A different shortcut strategy can be run in a loop: at each iteration, two random configurations are sampled on the path, and the optimizer tries to connect $\mathbf{q}_s$ to the first, the first one to the second one, and the second one to $\mathbf{q}_g$. The local paths that are still collision-free are then kept to make a shorter path, as shown in Figure 1.3, right.

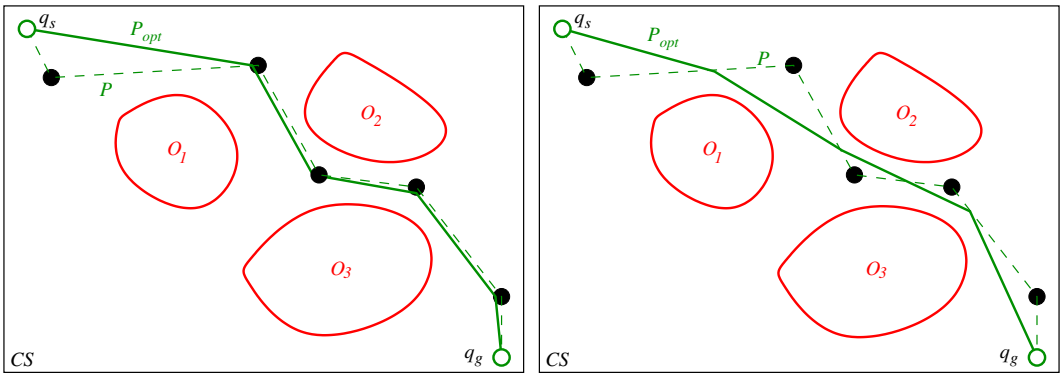


Figure 1.3: Left: The path P (in dashed green) is optimized with a greedy optimizer. Right: The optimized path P_{opt} (in continuous green) after several iterations of random optimization (RO).

1.2 Anthropomorphic Systems

We focus in this work on humanoid robots and digital actors, which are anthropomorphic systems. Such systems have a high number of DoF, and are capable of accomplishing human-like tasks: locomotion (such as walking, running and parkour), manipulation, or both. These tasks can be accomplished thanks to the fact that anthropomorphic systems are both underactuated and highly redundant.

In the remainder of this work, we will refer indistinguishably to anthropomorphic systems, humanoid robots and digital actors.

1.2.1 Underactuated Systems

A robot R is usually composed of a set of rigid bodies $(B_i)_i$, $i \in 0..N_B$, and a set of joints $(J_i)_i$, $i \in 1..N_J$, which constrain the body positions. A rigid body has a mass, an inertia and a given geometry. We use the kinematic tree formalism proposed in [Featherstone 08] to model a full robot: a node of the tree represents a body of the robot, while an arc (or edge) represents a joint J_i of the robot which constrains the motion of the successor body B_i with respect to its parent body B_{λ_i} (see Figure 1.4). Note that in the tree representation, each body has one and exactly one parent body, except for the root body which has no parents; this means that additional constraints have to be added later on in order to correctly model robots with closed kinematic chains such as parallel robots, or humanoid robots when they are in contact with their environment.

Joints usually correspond to the actuators on the physical robot. Each type of actuator will then have an equivalent type of joint (prismatic, revolute, etc). A configuration $\mathbf{q}$ of such a robot can then be defined as the concatenation of all joint DoF values, and the set of all configurations is called the actuated configuration space $\mathcal{Q}$. This is however not sufficient in the particular case of anthropomorphic systems, which rely on making and breaking contact with their environment – the

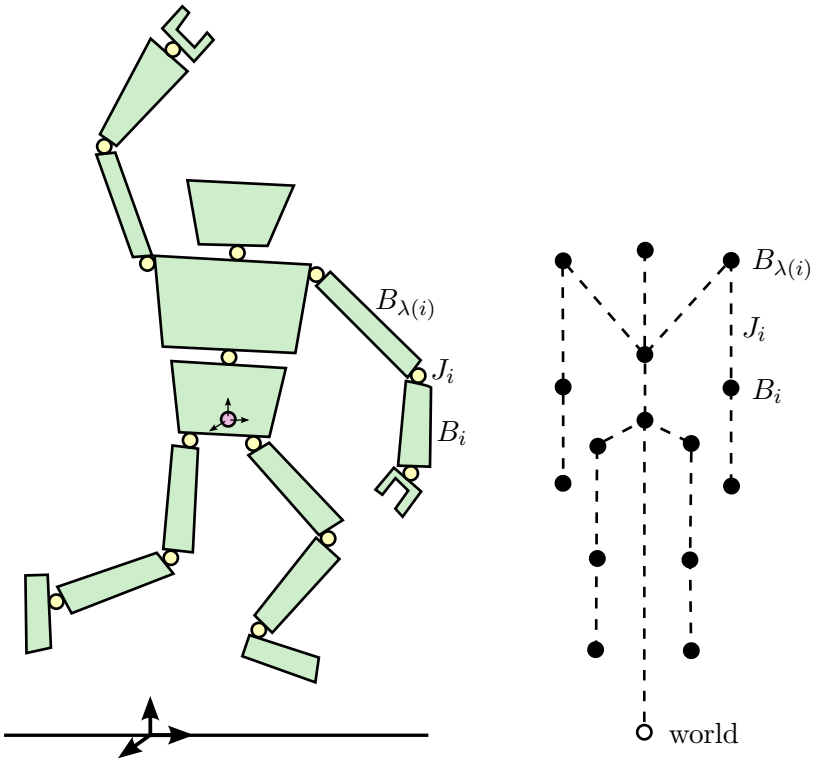


Figure 1.4: Left: schematic view of a humanoid robot: bodies are connected with joints (yellow circles) which represent the actuators. A fictitious 6-DoF joint, or floating joint (purple circle), is added to move the robot in $SE(3)$. Right: A kinematic tree view of the same robot, where bodies and joints are represented by nodes and edges respectively.

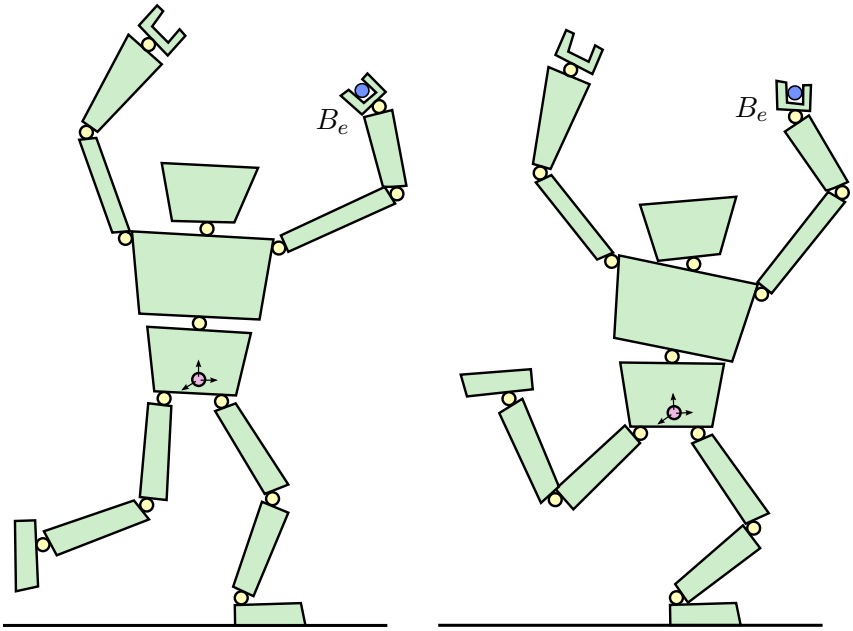


Figure 1.5: Anthropomorphic systems are highly redundant systems. For a desired Cartesian position (in blue) of the end effector B_e, there exists more than one configuration $\mathbf{q}$ that accomplish this task. Left and right: two possible solution configurations.

floor for instance – in order to move in their workspace. Additional information in the configuration vector is needed to model the general position of the system, and not only its actuators. Anthropomorphic systems are therefore said to be *underactuated systems*.

We therefore introduce a 6-DoF floating joint, which we attach to the root of the existing kinematic tree containing the actuated joints. The successor body of the floating joint will be called the floating base. Note that any body of the kinematic tree can be chosen to be the floating base; in Figure 1.4, the floating base is the "waist" of the robot. Thus, a full configuration $\mathbf{q}$ of the robot R is an element of the configuration space $\mathcal{CS} = SE(3) \times \mathcal{Q}$.

1.2.2 Kinematic Redundancy

We briefly introduce here the concept of kinematic redundancy. Figure 1.5 shows that for a same target Cartesian position of the end effector B_e, there exists more than one configuration of R that allows B_e to reach the target. The robot R is therefore kinematically redundant with respect to the task of reaching the object. One could imagine assigning multiple tasks to be accomplished at the same time, or even exploring $\mathcal{CS}$ while continuously accomplishing one or more tasks. This will be discussed more thoroughly in Chapter 2.

1.3 Walking and Balance

As stated in Section 1.2.1, anthropomorphic systems are underactuated. This means that they have to press some of their bodies against their environment in order to produce a displacement. In terms of dynamics, this is equivalent to saying that the environment exerts forces on contact surfaces of the robot. Note, however, that not all kinds of forces can be achieved, as the environment cannot for instance pull a body towards it in order to retain a contact. Great care must be then given to make sure that a planned motion is indeed achieved on a given system and environment. The concept of *balance* can be thus introduced: a motion will be balanced, provided that there are sufficient external forces which allow both achieving this motion and keeping the system physical integrity. In the general case, this does not necessarily imply that the system will not fall; a humanoid robot executing a back-flip is technically falling in the absence of contact forces, but, as long as it lands back safely on the ground without damaging its bodies, sensors or actuators, this motion is deemed to be balanced.

In this work, as we focus on the particular case of humanoid walking on a flat floor, we can rely on results from walking system stability analysis [Wieber 02] in order to verify and guarantee an anthropomorphic system balance during a walking motion.

1.3.1 Zero-Moment Point (ZMP)

We assume here that the robot R is walking on a flat horizontal floor, to which we associate the normal vector $\mathbf{u}$. We also assume that the robot is always in contact with the floor using its feet, i.e. that there are no jumps, and that all contacts are non-sliding. We will call these assumptions the *walking conditions*. R is then subject to its weight and to contact forces, and the total wrench of applied forces can be written as:

$$\begin{pmatrix} \sum_i m_i \mathbf{g} + \sum_k \mathbf{f}_{c_k} \\ \sum_i m_i \mathbf{x}_i \times \mathbf{g} + \sum_k \mathbf{p}_{c_k} \times \mathbf{f}_{c_k} \end{pmatrix}, \tag{1.1}$$

where m_i and $\mathbf{x}_i$ are respectively the mass and center of mass vector of rigid body B_i, $\mathbf{g}$ is the gravity acceleration vector, and $\mathbf{p}_{c_k}$ and $\mathbf{f}_{c_k}$ are respectively the position and force vectors for contact c_k.

Newton's second law of motion states that the total wrench of forces must be equal to the system dynamic wrench, denoted $\begin{pmatrix} \mathbf{f} \\ \mathbf{n} \end{pmatrix}$. Let m and $\mathbf{x}_G$ denote respectively the total mass of the robot and its *Center of Mass (CoM)* position. Assuming that the walking conditions are verified, the analysis presented in [Wieber 02] can be applied to guarantee that a walking motion is *dynamically balanced* if and only if the vertical projection of the point defined by:

$$\frac{m\mathbf{g}\mathbf{x}_G + \mathbf{u} \times \mathbf{f}}{m\mathbf{g} + \mathbf{f}.\mathbf{u}} \tag{1.2}$$

is always in the interior of the convex hull of the contact points; the convex hull is also known as the *support polygon*.

This point is called the *Zero-Moment Point (ZMP)* [Vukobratovic 69], where it is defined as the point on the floor where the horizontal components of the total moment are zero. It is also known as the *Center of Pressure (CoP)*, as it can be found by computing the barycenter of contact points weighted by normal contact forces applied on them. We further discuss the use of the ZMP formulation in Chapter 3.

Note that in the particular case where the dynamic wrench is zero, i.e. when body velocities and accelerations are zero, the point defined in Equation (1.2) coincides with the CoM. This allows us to guarantee that a quasi-static walking motion is *statically balanced* if and only if the vertical projection of the CoM is always in the interior of the support polygon.

1.3.2 Cart-Table Model

The above formulation of ZMP is rather hard to control. One way to deal with the complexity of a humanoid robot kinematic tree is to use the so-called "cart-table" simplified model, where the walking robot is modeled by a point mass at a fixed height, see Figure 1.6. The equations giving the ZMP horizontal coordinates (p_x, p_y) as functions of the CoM horizontal coordinates (x, y) in the cart-table model were presented in [Kajita 03]:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x - \frac{z_c}{g}\ddot{x} \\ y - \frac{z_c}{g}\ddot{y} \end{pmatrix} \tag{1.3}$$

where z_c is the constant height of the CoM and g is the gravity constant.

Based on this simplified model, a walking pattern generator is proposed in [Kajita 03]. Starting from a time-parameterized footprint sequence, a reference ZMP trajectory is built such that it connects the footprint centers and remains inside the support polygon. The support polygon consists of one footprint in single-support phases, and of the convex hull of two consecutive footprints in double-support phases. A preview-control method uses then Equation (1.3) to derive the CoM trajectory, see Figure 1.7, and inverse-kinematics methods are finally used to generate a whole-body dynamically balanced walking motion of the system.

1.4 Humanoid Walk Planning

The motion planning problem is certainly a complex one in the case of humanoid robots, which are high-DoF redundant systems that have to verify bipedal balance constraints. Various planning strategies can be found in the literature.

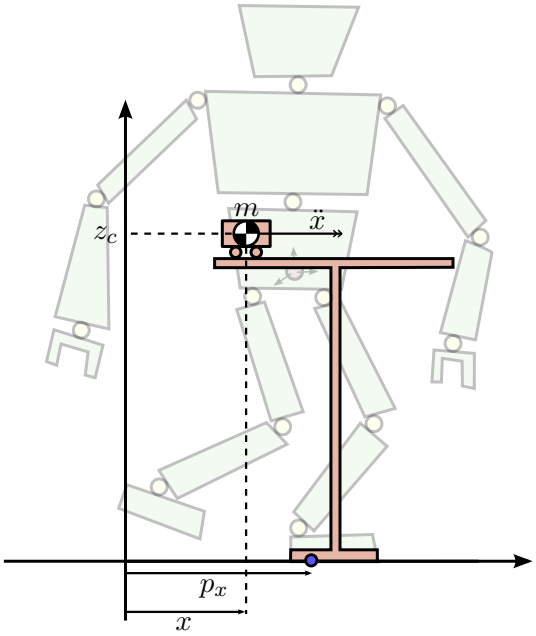


Figure 1.6: Simplified model of a cart on table. The cart can move horizontally along one dimension on the table with a non-zero acceleration and represents the robot CoM. The table foot represents the contact surface of the robot. Note that in its current state, the robot is dynamically balanced, as the ZMP lies on the table foot. The CoM vertical projection, on the other hand, is outside the table foot and static balance is not ensured.

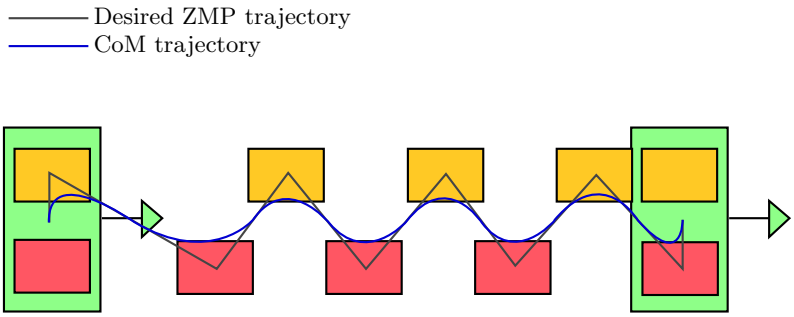


Figure 1.7: An illustration of the walking pattern generator: a desired ZMP trajectory (in grey) connects footprints. During the single-support phase, the ZMP stays under the support foot, and switches feet during the double support phase. The cart-table model is then used to produce the corresponding CoM trajectory (in blue).

1.4.1 Footstep Planning

One possible way of addressing humanoid walk planning is by reasoning on the footstep level. In [Kuffner 01, Chestnutt 05], humanoid footstep planning schemes are described. Starting from an initial footstep placement, they use an A* graph search [Hart 68] to explore a discrete set of footstep transitions. The search stops when the neighborhood of the goal footstep placement is reached. A similar approach [Garimort 11] that uses D* Lite allows fast re-planning in the presence of dynamic obstacles. While the previous methods are very efficient, they are not practical in environments with narrow passages; in [Xia 09, Perrin 12b], the computational cost of footstep planning is reduced by adapting RRT planning algorithm and efficiently exploring the discrete footstep space. In their most recent work, [Perrin 11, Perrin 12a] give an elegant proof of the equivalence between the discrete footstep planning problem and a continuous motion planning problem, which enables the use of off-the-shelf planning algorithms such as RRT.

1.4.2 Constraints-Based Motion Generation

Another category relies on prioritized whole-body task planning: kinematic redundancy is used to accomplish tasks with different orders of priorities [Khatib 04, Saab 12]. Dynamic balance and obstacle avoidance can then be defined as unilateral constraints that the algorithm has to verify during the whole motion. The trajectory is then generated by specifying a set of goal tasks – which could be a goal configuration – and let this set act as an attractor, in a way similar to the simpler example shown in 1.1.1. Using such as scheme, the work in [Kanoun 09] defines a robot augmented with a sequence of footprints and formulates the problem of locomotion planning as an optimization problem. Such a scheme works well in the presence of simple obstacles, but is prone to falling in local minima, failing to find solutions for more complex environments.

1.4.3 Constrained Motion Planning

More recently, sampling-based motion planning algorithms were adapted to efficiently explore a constraint submanifold of $\mathcal{CS}$. In [Bretl 06, Hauser 10a], this strategy is used to plan on a union of submanifolds, where each manifold is defined by a contact limb position and static balance constraints, producing statically balanced locomotions for hexapods and humanoid robots on uneven terrain. Constrained motion planning will be further discussed in Chapter 2, where a new dynamically-balanced locomotion planner is introduced.

1.4.4 Multi-Contact Planning

It is interesting to note that while collision avoidance is a requirement in motion planning, some features of the environment can in fact be useful to solve the problem, especially in the case of underactuated systems. In [Bouyarmane 12, Escande 13], a

multiple-contact-point stance planner looks for authorized contact surfaces in order to help the robot reach its goal. The contact points do not have to be coplanar, and the planner produces a sequence of statically balanced postures for a humanoid robot which can press his feet and hands against objects in the environment. A heuristic method is used to direct the search towards the goal, so this planner does not offer a completeness property. Once the contact stance sequence is obtained, motion generation tools can be used to generate statically balanced or dynamically balanced locomotion trajectories.

1.4.5 Decoupled Planning

Finally, another strategy consists in dividing a high-dimensional problem into smaller problems and solving them successively [Zhang 09]. The idea of dividing the problem into a two-stage scheme is described in [Yoshida 08]: A 36-DoF humanoid robot is reduced to a 3-DoF bounding box. Using the robot simplified model, the PRM algorithm solves the path planning problem and generates a valid path for the bounding box. A geometric decomposition of the path places footsteps on it, and the walk pattern generator described in Section 1.3 finally produces the whole-body trajectory for the robot. In [Moulard 10], this two-stage approach is also used; numerical optimization of the bounding box path produces a time-optimal trajectory that is constrained by foot speed and distance to obstacles. Optimization techniques will be discussed in details in Chapter 3, where we describe an optimal motion planning framework.

1.4.6 Holonomic vs Nonholonomic Walking Motion

An important notion on humanoid walk planning is the one of holonomic motion. A *holonomic constraint* is given by:

$$C(\mathbf{q}) \quad = \quad 0, \tag{1.4}$$

with $\mathbf{q}$ the configuration vector. When a constraint of the form

$$C(\mathbf{q},\dot{\mathbf{q}}) \quad = \quad 0, \tag{1.5}$$

with $\dot{\mathbf{q}} = \dfrac{d\mathbf{q}}{dt}$, cannot be integrated into a form similar to (1.4), this constraint is called *nonholonomic*.

In practice, a nonholonomic constraint implies that a system velocity vector cannot take an arbitrary direction. For instance, you cannot drive a car sideways (this makes life harder for us when parking), and therefore it has a nonholonomic constraint. On the other hand, a spherical ball can roll in any direction, which means it has a holonomic constraint.

So is there a particular constraint that governs human gait? Studies, like the one described in [Mombaur 10], established a model that presents trajectory planning as an optimization problem of a cost function. Basically, it is shown that for small

distances and small orientation variations between the start and goal configurations, humanoid motion obeys a holonomic constraint, and sidestepping is allowed. However, for greater distances and orientation variations, a nonholonomic constraint rules human gait, and sidestepping is forbidden, i.e. the human direction is always tangent to its path. But one should note that this model is only applicable in the case of the absence of any obstacles, and it is clear that in the presence of narrow passages, a human would be forced to adopt holonomic motion in order to walk sideways.

The path planning scheme in [Yoshida 08] is designed to this end; a PRM algorithm first builds a roadmap with Dubins curves [Dubins 57], but such curves impose a nonholonomic constraint and narrow passages cannot be crossed. The roadmap is therefore enriched with linear local paths. As a result, this planning scheme generates motions such that the robot remains tangent to its path most of the time and uses sidestepping only in narrow passages.

1.5 Contribution: Regular Sampling Optimization

The work of [Moulard 10] provides a sound way for generating nice walking trajectories, using numerical optimization to minimize the robot walking time along the path while enforcing velocity and obstacle distance constraints. However, after having tried this approach, we came to the conclusion that it was too computationally expensive given the simple nature of the robot model, with optimizations taking several minutes even for simple environments.

While using the same two-stage approach of [Yoshida 08], a simpler heuristic method that generates near time-optimal humanoid trajectories is proposed. First the PRM algorithm and the Dubins local paths are replaced with an RRT-Connect algorithm and linear local paths. The path is then optimized by locally reorienting the robot bounding box on a discrete set of configurations. Our heuristic gives priority to nonholonomic motion; holonomic motion is used only to pass in narrow passages or avoid nearby obstacles, thus generating short walking trajectories.

The following section presents this method and explains how it is integrated in the motion planning scheme. Examples of different scenarios, including a real one with the HRP-2 platform, are shown in section 1.7.

1.6 Regular Sampling Optimization

Assuming full knowledge of the environment, the RRT algorithm produces a collision-free piecewise linear path P_{RRT} for the robot bounding box (in offline mode), i.e. the path consists of the concatenation of linear local paths LP_{RRT}.

Due to the probabilistic nature of RRT, P_{RRT} may not be optimal in terms of length, and a preliminary random shortcut optimization (RO) can be run in order to shorten it (See Figure 1.8). Though the optimized path P is collision-free, the

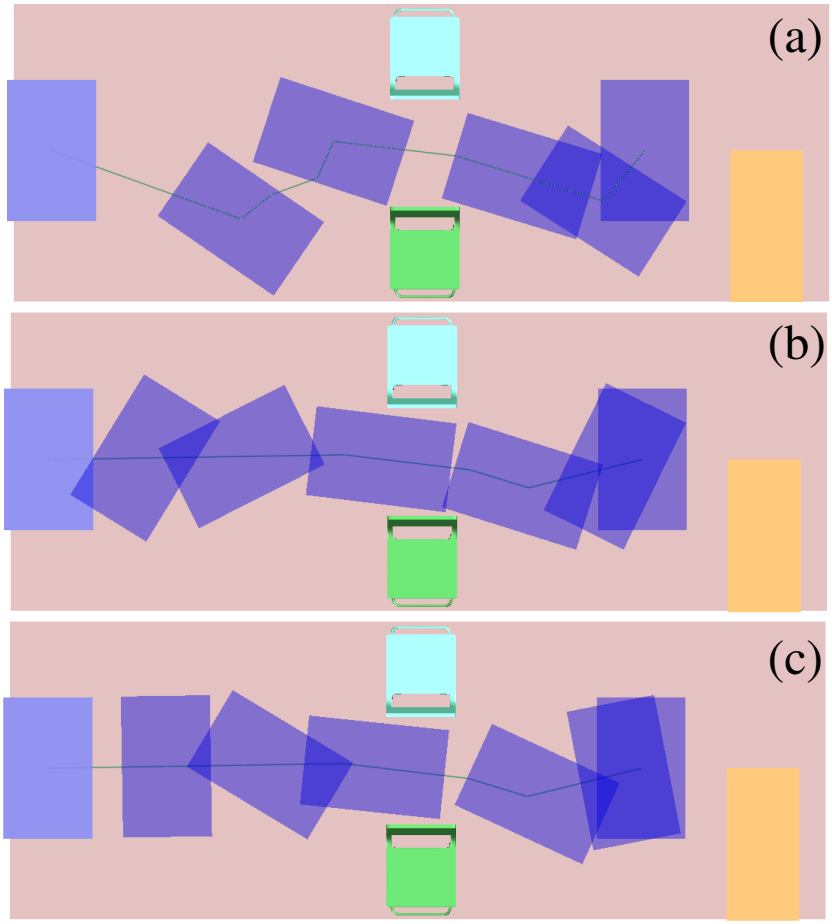


Figure 1.8: Top view: (a) RRT-Connect path for the bounding box passing between two chairs. (b) Optimized bounding box path by random optimization (RO). (c) Optimized bounding box after adding regular sampling optimization (RSO) .

bounding box orientation is such that it could lead to a trajectory which is not time-optimal. For instance, the humanoid robot could spend a long time walking sideways or backwards over a long distance in an open space. An additional optimization stage is introduced to address this issue in the next section.

1.6.1 Bounding Box Path Optimization

Note that each configuration $\mathbf{q}$ can be written as $\mathbf{q} = (\mathbf{X},\ \theta)$, where $\mathbf{X} = (x,\ y)$ describes the bounding box position in the horizontal plane, and θ gives its orientation. The optimizer reorients the bounding box along P by changing θ while retaining the value of $\mathbf{X}$.

For this purpose, an A* search algorithm is executed. First, P is regularly sampled. Using a discrete set of admissible orientations for each sample configuration, a cost function and a heuristic estimation function, the bounding box orientation is then modified along P. An optimized path P_{opt} is created and leads to a trajectory which is time-optimal with the respect to the expanded graph.

Preliminary Notations

After running RO on the piecewise linear path P_{RRT}, the path P is also piecewise linear, and its first and last configurations are denoted by $\mathbf{q}_s$ and $\mathbf{q}_g$.

Let $d_{sample} \in \mathbb{R}^+ \setminus \{0\}$ be a sampling distance. Sampling P with a distance d_{sample} means dividing each local path LP_j of P into smaller local paths of length d_{sample}; each new local path end is then a sample configuration. Note that the last interval on each local path LP_j can have a length smaller than d_{sample}. The n^{th} sample configuration of P in its initial state can be obtained by indexing new local path ends starting from $\mathbf{q}_s$, and is denoted $\mathbf{q}_n^{init}$.

The admissible orientation states need to be defined. We aim to make a humanoid robot reach its goal as soon as possible. Since the robot is faster while walking straight than side-stepping, we attempt to change the orientation of each initial sample configuration $\mathbf{q}_n^{init}$ such that the bounding box is tangent to the local path, and we introduce a new configuration denoted $\mathbf{q}_n^{front}$. To take into account the fact that there may be obstacles that forbid a frontal orientation, we also create $\mathbf{q}_n^{lat_1}$ and $\mathbf{q}_n^{lat_2}$ that are rotated by $\frac{\pi}{2}$ and $-\frac{\pi}{2}$ relative to the path tangent, see Figure 1.9. One particular case is local path end configurations: the mean direction of the two adjacent local paths is considered to define frontal and lateral configurations. This is done to ensure a smooth transition between two local paths.

A sample configuration whose orientation is unknown will be denoted by $\mathbf{q}_n^{state}$. It can have any orientation state of the set $\{init,\ front,\ lat_1,\ lat_2\}$ except for $\mathbf{q}_s$ and $\mathbf{q}_g$ which remain in their initial state. Ideally, the algorithm should be able (as long as there are no obstacles) to put each sample configuration in the frontal state, create a new path P_{opt} and generate a time-optimal trajectory for the robot.

An A* search is run to achieve this goal; the algorithm functions are described in the following section.

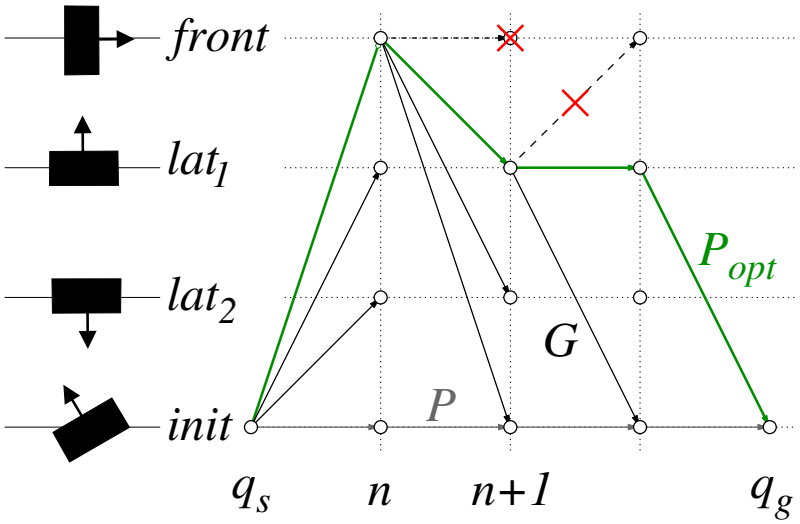


Figure 1.9: Each initial sample configuration can be rotated and be in one of four states. Starting from $\mathbf{q}_s$, the A* search algorithm searches the graph G that contains only valid nodes and arcs to produce an optimized path P_{opt}.

A* Function Definition

An A* search algorithm can find an optimal path in a graph as long as the latter and an evaluation function are correctly defined. Starting from $\mathbf{q}_s$, A* expands in each iteration the admissible transitions from one sample to the next one in the graph and evaluates, using the evaluation function, a lower bound of the cost-to-go from each state to the goal state, see Figure 1.9.

A graph G is defined to be a set of nodes and arcs. A valid node $\mathbf{q}_n^{state_n}$ is defined to be a configuration with no collisions, and a valid arc $\mathbf{q}_n^{state_n}\mathbf{q}_{n+1}^{state_{n+1}}$ is a collision-free local path. The whole graph G could be built before running A* by testing all nodes and arcs and making sure they are collision-free. But collision tests are slow, and A* uses a heuristic estimation function to avoid going through all nodes. An empty graph G is thus initialized and nodes and arcs are built only when necessary. A successor operator needs to be defined for this purpose.

The Successor operator $\Gamma(\mathbf{q}_n^{state_n})$: Its value for any node $\mathbf{q}_n^{state_n}$ is a set $\{(\mathbf{q}_{n+1}^{state_{n+1}},\ c_{n,n+1})\}$, where $\mathbf{q}_{n+1}^{state_{n+1}}$ denotes a successor node, and $c_{n,n+1}$ is the cost of going from $\mathbf{q}_n^{state_n}$ to $\mathbf{q}_{n+1}^{state_{n+1}}$. The cost $c_{n,n+1}$ is defined to be the distance $D(\mathbf{q}_n^{state_n},\ \mathbf{q}_{n+1}^{state_{n+1}})$ between two nodes of G; it computes the walk time

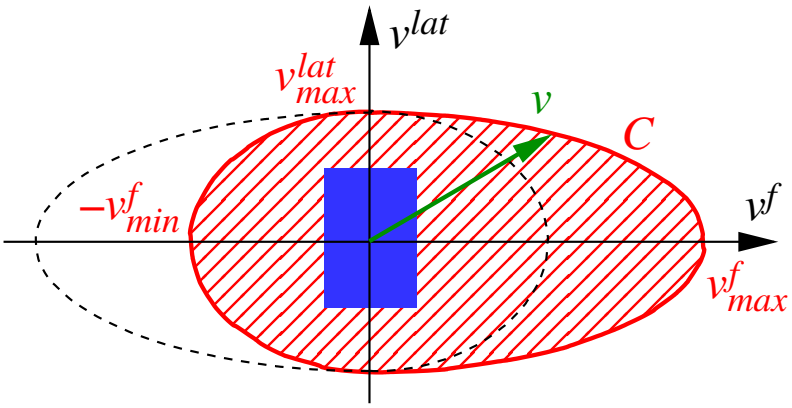


Figure 1.10: The rectangular bounding box speed vector v is bounded inside the hashed area defined by the speed constraint C. The area is bounded by the union of two half-ellipsoids.

from $\mathbf{q}_n^{state_n}$ to $\mathbf{q}_{n+1}^{state_{n+1}}$. The speed constraint C is defined as:

$$C = \begin{cases} (\frac{v^f}{v^f_{max}})^2 + (\frac{v^{lat}}{v^{lat}_{max}})^2 - 1 & \text{if } v^f >= 0 \\ (\frac{v^f}{v^f_{min}})^2 + (\frac{v^{lat}}{v^{lat}_{max}})^2 - 1 & \text{if } v^f < 0 \end{cases} \tag{1.6}$$

where v^f and v^{lat} are respectively the frontal and lateral speed, and v^f_{min}, v^f_{max} and v^{lat}_{max} their minimum and maximum values (See Figure 1.10). $D(\mathbf{q}_n^{state_n}, \mathbf{q}_{n+1}^{state_{n+1}})$ can be then computed by integrating this speed constraint along the linear path connecting $\mathbf{q}_n^{state_n}$ to $\mathbf{q}_{n+1}^{state_{n+1}}$.

Having expressed the successor operator, which allows the optimizer to choose which node to expand at each iteration, the A* evaluation function can be defined.

The Evaluation Function $\hat{f}(\mathbf{q}_n^{state})$: It is the estimated cost of an optimal path going through $\mathbf{q}_n^{state}$ from $\mathbf{q}_s$ to $\mathbf{q}_g$ and can be written as:

$$\hat{f}(\mathbf{q}_n^{state}) = \hat{g}(\mathbf{q}_n^{state}) + \hat{h}(\mathbf{q}_n^{state}) \tag{1.7}$$

where $\hat{g}(\mathbf{q}_n^{state})$ is the estimated cost of the optimal path from $\mathbf{q}_s$ to $\mathbf{q}_n^{state}$ and $\hat{h}(\mathbf{q}_n^{state})$ is a heuristic function giving the estimated cost of the optimal path from $\mathbf{q}_n^{state}$ to $\mathbf{q}_g$.

$\hat{h}(\mathbf{q})$ must verify $\hat{h}(\mathbf{q}) \leq h(\mathbf{q})$ for any $\mathbf{q}$ to ensure that the algorithm is admissible, i.e. that the path from $\mathbf{q}_s$ to $\mathbf{q}_g$ is optimal. Since the robot is fastest while walking

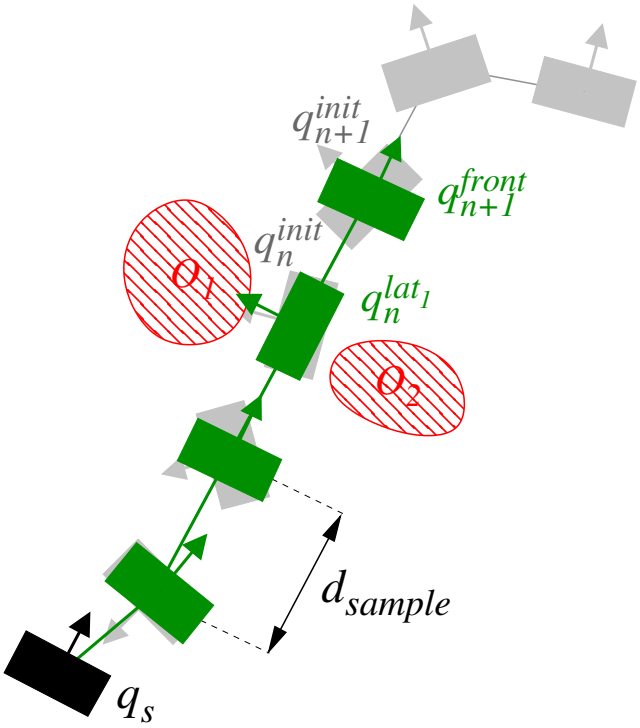


Figure 1.11: Local paths are regularly sampled (light grey) and each sample configuration is reoriented (dark) while considering obstacles O_1 and O_2.

straight forward in the absence of obstacles, $\hat{h}(\mathbf{q}_n^{state})$ is defined as:

$$\begin{aligned}\hat{h}(\mathbf{q}_n^{state}) = & D(\mathbf{q}_n^{state},\ \mathbf{q}_{n+1}^{front}) \\ & + \sum_{k=1}^{N_{sample}-n-2} D(\mathbf{q}_{n+k}^{front},\ \mathbf{q}_{n+k+1}^{front}) \\ & + D(\mathbf{q}_{n+1}^{front},\ \mathbf{q}_g)\end{aligned} \tag{1.8}$$

where N_{sample} is the total number of initial sample configurations in P including $\mathbf{q}_s$ and $\mathbf{q}_g$. $\hat{h}(\mathbf{q}_n^{state})$ thus sums the cost of walking along P while staying tangential to the path with the start and end transition costs from $\mathbf{q}_n^{state}$ and to $\mathbf{q}_g$.

Now that the A* functions are fully defined, a search algorithm can be run to compute an optimal path P_{opt} by changing the orientation of each sample node. Algorithm 2 describes the *Regular Sampling Optimization (RSO)* method, and an example is shown in Figure 1.11.

1.6.2 Motion Generation for a Humanoid Robot

A collision-free path P for the 3-DoF bounding box can be found using RRT-Connect and RO. The regular sampling optimization (RSO), which is the subject of this

Algorithm 2 RSO(P, d_{sample})

// Closed set
$\mathcal{C} \leftarrow \emptyset$
// Open set
$\mathcal{O} \leftarrow \{(\mathbf{q}_s, \hat{f}(\mathbf{q}_s))\}$
$\mathbf{q}_n^{state_n} \leftarrow \mathbf{q}_s$
while $\mathbf{q}_n^{state_n} \neq \mathbf{q}_g$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mathbf{q}_n^{state_n}, \hat{f}(\mathbf{q}_n^{state_n}))\}$
  // Expand node
  $\mathcal{E}_{n+1} \leftarrow \{(\mathbf{q}_{n+1}^{state_{n+1}},\ c_{n,n+1})\} \leftarrow \text{Expand}(\mathbf{q}_n^{state_n}, P, d_{sample})$
  for $\mathbf{q}_{n+1}^{state_{n+1}} \in \mathcal{E}_{n+1}$ **do**
    // Mark as "open" each successor not already marked "closed"
    if $(\mathbf{q}_{n+1}^{state_{n+1}}, \hat{f}) \notin \mathcal{C}$ **then**
      $\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathbf{q}_{n+1}^{state_{n+1}}, \hat{f}(\mathbf{q}_{n+1}^{state_{n+1}})\}$
    else
      // Remark as open each closed successor for which evaluation function is now smaller than stored value
      if $\hat{f}(\mathbf{q}_{n+1}^{state_{n+1}}) < \hat{f}$ **then**
        $\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathbf{q}_{n+1}^{state_{n+1}}, \hat{f}(\mathbf{q}_{n+1}^{state_{n+1}})\}$
      end if
    end if
  end for
  // Select open node whose value evaluation function value is the smallest
  $\mathbf{q}_n^{state_n} \leftarrow \arg\min_{\mathbf{q} \in \mathcal{O}} \hat{f}(\mathbf{q})$
end while
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mathbf{q}_g, 0)\}$
// Return closed set which makes up optimized path.
return $\mathcal{C}$

	RRT-Connect	RO	RSO	Robot Trajectory	Total
Chairs	3.97	1.89	2.14	66.1	74.1
Boxes	0.0917	2.50	0.238	65.7	68.6
Apartment	1.21	2.43	2.41	223	229

Table 1.1: Computational time, in seconds, of each planning stage for the presented scenarios.

	RO	RO+RSO
Chairs	40	35
Boxes	66	57
Apartment	200	120

Table 1.2: Humanoid robot walk time, in seconds, for the presented scenarios using RO alone and a RO-RSO combination.

work, is then applied on the path and produces a path P_{opt} that gives priority to nonholonomic motion.

Once the bounding box trajectory is computed, the robot has to walk along it. A footstep sequence is thus generated along P_{opt} by geometric decomposition of the path, and the pattern generator cited in Section 1.4 then produces the robot whole-body trajectory.

1.7 Examples

This section presents experimental results of the path optimizer after it has been inserted in the previously described walk planning scheme. Distance parameters v^f_{max}, v^{lat}_{max}, v^f_{min} are set to 0.5, 0.1, and 0.25 $m.s^{-1}$ respectively. Note that these parameters play a key role in giving priority to forward walking with respect to lateral and backwards walking. Indeed kinematic constraints are usually such that it is very hard to achieve fast lateral walking. Also sensors, such as the cameras facing the forward direction, can require preferring forward walking to backward walking.

Since the A* search only takes place over the graph of discretized configurations of the input path, it is important to choose a proper value of the sampling interval d_{sample}. If the value is too small, the search space will be much bigger, possibly leading to better trajectories, but leading to an explosion of the A* search time. If the value is too high, the A* search will be very fast in the small trajectory space, but we risk missing some optimizations on the trajectory. We found that setting d_{sample} to be equal to $\frac{h}{6}$, where h is the humanoid height, provided a good tradeoff between computation time and trajectory quality; d_{sample} gives then a broad approximation of the nominal human step length.

Tests are performed on a 2.13 GHz Intel Core 2 Duo PC with 2 GB RAM.

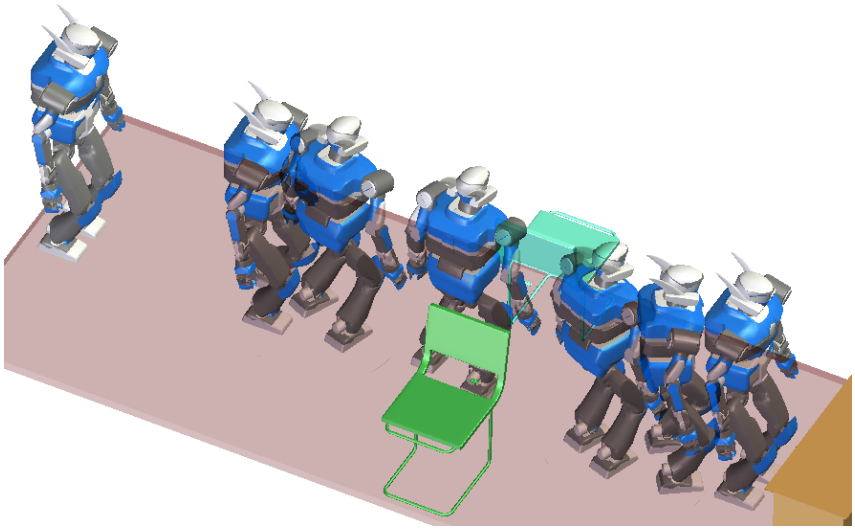


Figure 1.12: Perspective view of the simulated HRP-2 trajectory on the final optimized path passing between two chairs.

Simulations of the humanoid robot HRP-2 are run in three scenarios. The first one is a small environment where HRP-2 has to pass between two chairs. The second environment is uncluttered with few obstacles lying around, while the last one is a bigger apartment environment where the robot has to move from one room to another while passing through doors. The chairs scenario motion is also replayed on the real humanoid robot HRP-2 [Kaneko 04], see Figure 1.13.

Table 1.1 shows computation times for each stage of the planning scheme: RRT-Connect, RO, RSO, and the whole-body robot trajectory generation. In order to show the optimizer contribution, robot walk times are also measured by creating a trajectory directly after RO, and comparing it with a trajectory where the RSO was added, see Table 1.2.

1.7.1 "Chairs" Scenario

Figure 1.8 shows the bounding box RRT path and the RO path for the chairs scenario. It is obvious that RO creates a shorter path. However, the bounding box starts rotating from the beginning of the path even though both chairs are still far. This causes the robot trajectory to not be time-optimal since walking sideways takes a longer time than walking straight.

However, after applying RSO, it is clear that the bounding box stays oriented towards the front and rotates only when it reaches the chairs. Figure 1.12 and Table 1.2 show that the walk time is 12% shorter and the final trajectory for HRP-2 is more realistic. Note that the RSO takes 2,144 ms to be executed on the chairs path, which is less than 3% of the total computation time.

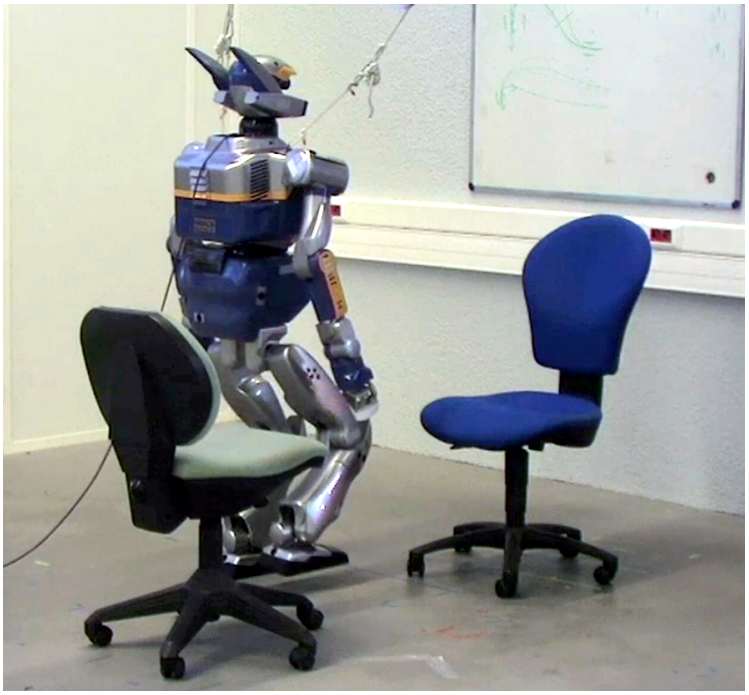


Figure 1.13: Humanoid Robot HRP-2 uses holonomic motion, or side-stepping, to pass between two chairs.

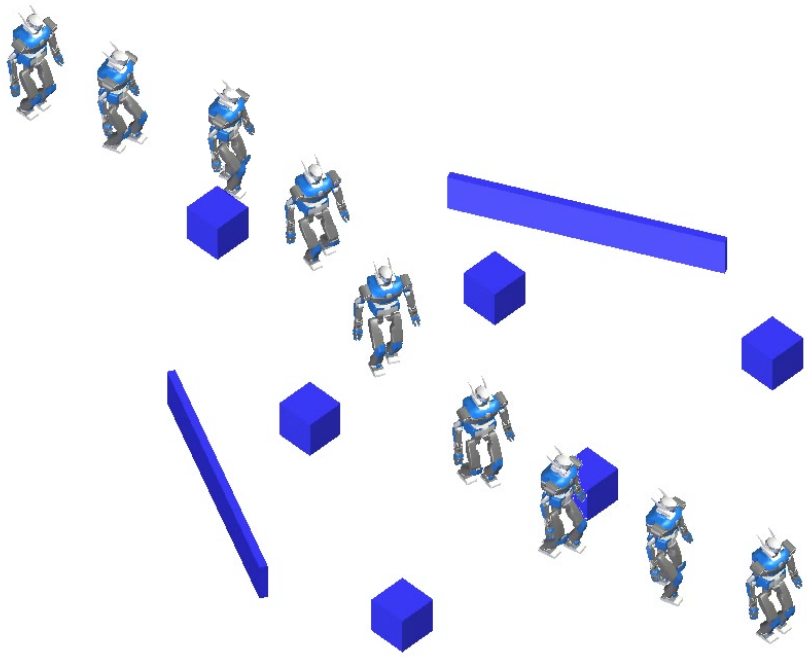


Figure 1.14: Perspective view of HRP-2 optimized trajectory in the boxes scenario.

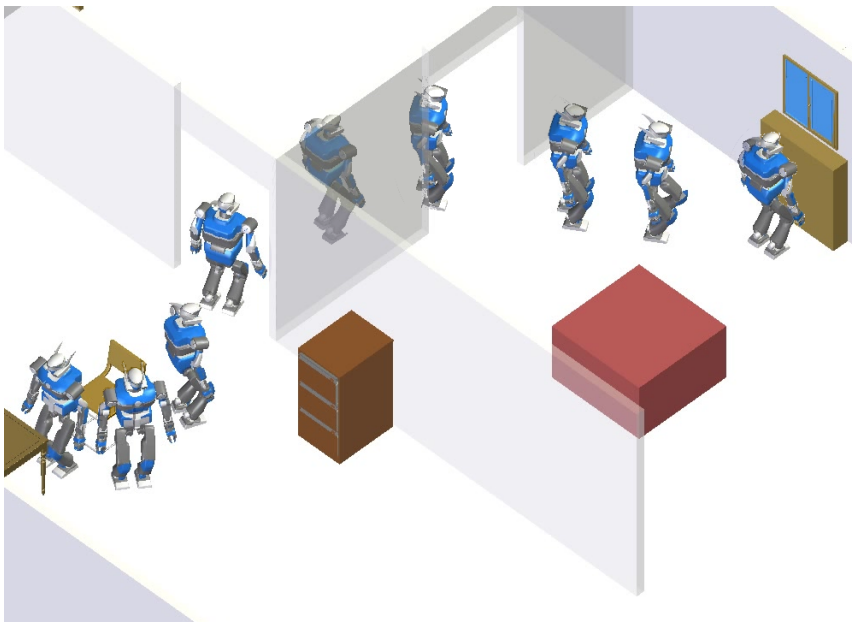


Figure 1.15: Perspective view of HRP-2 optimized trajectory in the apartment scenario.

1.7.2 "Boxes" Scenario

Here, an uncluttered environment is considered, and it can be seen that RRT-Connect and RSO computation times are very low compared to other environments. This can be explained by the fact that a tree connecting start and goal configurations is easier to find, and that the frontal orientation state is valid for all considered samples on the path, see Figure 1.14.

1.7.3 "Apartment" Scenario

The planning scheme is finally applied in the apartment scenario. In Figure 1.15, it is evident that HRP-2 walks facing forward through the doors. As with previous scenarios, the trajectory is more realistic than a trajectory where RSO is not used. The added computation time for RSO is 2,412 ms, which is insignificant compared to the 228 seconds which are required by the whole planning scheme.

Additionally, since the environment is significantly larger and more constrained than the previous ones, the walk time difference is more striking: Table 1.2 shows that it takes the robot 40% less time to cross the apartment when an RO-RSO combination is used.

1.8 Conclusion

In this chapter, a novel simple optimization method is presented for humanoid walk planning that relies on a decoupling between trajectory and robot orientations. It

uses an A* search that takes as input a path for the robot bounding box, and produces a path where a discrete set of configurations have been reoriented to generate a realistic time-optimal walk trajectory. Results show that new trajectories are more satisfactory while the added computation time is insignificant compared to the whole planning time.

Achieving humanoid walk planning on flat surfaces using the bounding box approach is both simple and efficient; it allows using off-the-shelf sampling-based planners to efficiently explore the configurations space, thus removing the need to take into account additional kinematic and balance constraints when planning for the whole articulated robot. But it has one major drawback: in order to have a sound framework which always produces collision-free walking trajectories, the bounding box must contain all the robot geometries and take into account the robot swaying motion during walking. Thus, it is a rather conservative method which cannot be used for manipulation tasks, and which might not even succeed in finding collision-free walk trajectories in cluttered environments. If we go back to the chairs example in Section 1.7.1, we can show that the lateral walking motion could have been avoided if the HRP-2 had lifted its arms over the two chairs. Beside leading to an even faster motion, walking forward could be necessary if vision systems were needed to achieve robot localization and/or environment mapping.

In the next chapter, we introduce a sound whole-body motion planning algorithm that allows planning collision-free walking trajectories for the fully articulated humanoid robot, hence releasing it from its bounding box constraint and increasing its accessible workspace.

Chapter 2

Dynamic Walking and Whole-Body Motion Planning for Humanoid Robots: an Integrated Approach

The humanoid walk motion planning problem was tackled in Chapter 1 by first planning a geometric path for the robot bounding box, then transforming the path into a dynamic walking trajectory by laying footprints along it and using a pattern generator. However, this approach reasons only on the walking level, and does not take into consideration the fact that the upper body could move. Indeed, some difficult and complex situations may require considering exact 3D models of a humanoid robot and its environment. This applies when passing between two chairs for instance (see Section 1.7.1), as lifting the arms of the robot outside of the bounding box could enable the humanoid robot to use forward walking during the whole motion, which will lead to faster execution time. Furthermore, as the robot arms are primarily used for manipulation, freeing them from the bounding box can allow the generation of whole-body motions involving both locomotion and manipulation.

This chapter presents a general method for planning collision-free whole-body walking motions for humanoid robots. We rely on a randomized algorithm for constrained motion planning; it is used to generate collision-free statically balanced paths solving manipulation tasks. Then, we show that dynamic walking makes a humanoid robot small-space controllable. Such a property allows to easily transform collision-free statically balanced paths into collision-free dynamically balanced trajectories. It leads to a sound algorithm which has been applied and evaluated on several problems where whole-body planning and walk are needed, and the results have been validated on the HRP-2 robot.

2.1 Motion Planning in Submanifolds of the Configuration Space

Sampling-based planners, such as the ones presented in Section 1.1.2, have encountered wide success in generating collision-free paths in high-dimension configuration spaces. When using sampling techniques on a humanoid robot, a major difficulty is to take into account contact and balance constraints, which is equivalent to generating random configurations on zero volume submanifolds of $\mathcal{CS}$. Indeed, the probability of sampling a configuration $\mathbf{q}_{rand}$ in $\mathcal{CS}$ such that it lies on such manifolds is zero. In this section, we present recent advances in motion planning on constraint manifolds using inverse-kinematics (IK) solvers.

2.1.1 Inverse Kinematics

The problem of inverse kinematics for a humanoid robot, or any articulated structure, is to compute a configuration $\mathbf{q}$ to achieve a task $\mathbf{x}$. This task is usually expressed in the Cartesian space, and can represent an end-effector position and/or orientation, the CoM position, etc. Some tasks may have more than one solution configuration, see Section 1.2.2. In the case of most robotics tasks for redundant systems, the set of solution configurations has a specific topological structure and forms a differentiable submanifold $\mathcal{M}$ of $\mathcal{CS}$, i.e. a set that locally "looks like" the euclidean space $\mathbb{R}^m$, and such that the tangent vector space $T_{\mathbf{q}}\mathcal{M}$ is defined for any $\mathbf{q}$. m is called the dimension of the manifold $\mathcal{M}$.

As the robots we deal with are redundant, it is natural to take advantage of this redundancy by specifying multiple tasks, potentially with different priorities. This problem has been widely studied in robotics planning and control literature, and many Jacobian-based solutions have been proposed, among which [Nakamura 86], [Siciliano 91], [Baerlocher 98], [Khatib 04] and [Kanoun 09]. Obstacle avoidance can be taken into account with similar methods. To do so, one has to include the obstacles as constraints to satisfy, see for example [Kanehiro 08]. These methods are prone to fall into local minima, thus global motion planning is needed to overcome this limitation. Note that when local methods find solutions, these are usually smoother; the choice of using global motion planners is justified by the need for complete algorithms.

We show here a functional example of an IK solver: its purpose is to find the root $\mathbf{q}$ of a non-linear C^1 function $f(q)$ with a tolerance of ϵ. If we want to find a configuration on a manifold $\mathcal{M}$, $\mathbf{f}(\mathbf{q})$ can be defined as a vector-valued function that contains the concatenation of all constraints defining $\mathcal{M}$. Note that as the intersection of two or more manifolds is also a manifold, this constraint solver allows us also to generate configurations that lie at the intersection of several manifolds.

Algorithm 3 implements a Newton-Raphson method [Bonnans 06]: starting from an initial value of $\mathbf{q}$, $\mathbf{q}$ is updated iteratively by $-\alpha \left(\frac{\partial \mathbf{f}}{\partial \mathbf{q}}(\mathbf{q})\right)^{+} \mathbf{f}(\mathbf{q})$, where α denotes

a gain and $\left(\frac{\partial \mathbf{f}}{\partial \mathbf{q}}(\mathbf{q})\right)^+$ denotes the Moore-Penrose pseudo-inverse of the Jacobian of $\mathbf{f}(\mathbf{q})$. The use of an adaptive gain α, which increases iteratively from an initial value α to a maximum value $\alpha_{max} \in [0, 1]$, allows the overshoot avoidance and convergence acceleration. The update rule relies on a real factor $w \in [0, 1]$; the lower w is, the faster α will reach α_{max}. Obviously, beside the solver parameters, the solver convergence depends of the initial value of $\mathbf{q}$, and a bad initialization can lead to either slow convergence or failure. A cutoff number of iterations it_{max} is thus introduced to bypass these cases. Note that in this simplistic implementation, joint limit position bounds are not enforced. Clamping the update at each iteration allows making sure they are not violated.

In practice, we observe that values of $\epsilon = 10^{-6}$, $\alpha = 0.1$, $\alpha_{max} = 0.95$ and $w = 0.8$ lead to good behavior, i.e. fast convergence and low failure rate. These values are kept constant for all scenarios in this work.

Algorithm 3 `SolveConstraints`($\mathbf{q}$, $\mathbf{f}$, ϵ): find $\mathbf{q}$ such that $\mathbf{f}(\mathbf{q}) = 0$

$i = 0$
while $\|\mathbf{f}(\mathbf{q})\| > \epsilon$ and $i \leq it_{max}$ **do**
  // $(.)^+$ denotes the Moore-Penrose pseudo-inverse
  $\mathbf{q} \leftarrow \mathbf{q} - \alpha \left(\frac{\partial \mathbf{f}}{\partial \mathbf{q}}(\mathbf{q})\right)^+ \mathbf{f}(\mathbf{q})$
  $i \leftarrow i + 1$
  // Make α tend toward α_{max}
  $\alpha \leftarrow \alpha_{max} - w(\alpha_{max} - \alpha)$
end while
if $\|\mathbf{f}(\mathbf{q})\| \leq \epsilon$ **then**
  return $\mathbf{q}$
else
  return failure
end if

2.1.2 Randomized Motion Planning on Constraint Manifolds

This problem of motion planning on constraint submanifolds of $\mathcal{CS}$ has been investigated with success during the last few years; the work of [Berenson 11] presents an exhaustive survey of Jacobian-based methods. Other recent contributions [Porta 12] present sophisticated constrained motion planning techniques based on higher-dimensional continuation.

This section presents an algorithm for constrained motion planning on a submanifold $\mathcal{M}$ of the configuration space $\mathcal{CS}$. It presents a simple adaptation of the RRT algorithm to constrained motion planning, that was first introduced in [Dalibard 09]. A configuration $\mathbf{q}$ of $\mathcal{CS}$ is said to be valid iff, beside being collision-free, it lies on the manifold $\mathcal{M}$ up to the tolerance ϵ; we call $\mathcal{M}$ the planning manifold.

The problem solved here differs from classic approaches in two ways:

1. the set of valid configurations is defined implicitly, as the set of collision-free configurations satisfying a given set of inverse kinematics balance constraints;
2. the goal manifold $\mathcal{M}_g$ is also defined implicitly, by additional inverse kinematics constraints.

During global planning, several types of constraints are considered for various reasons:

– Static balance: the CoM of the robot stays above the support polygon center, the two feet are horizontal on the ground.

– End-effector position and orientation: the goals of some problems presented in the experimental section of this chapter are defined as a specific robot hand pose, or a gaze direction.

– Configuration task: this adaptation of randomized motion planning algorithms uses tasks defined as the distance towards a given configuration in $\mathcal{CS}$. This will be detailed in the following section.

This formulation of manipulation planning does not include an explicit goal configuration, so it is not possible to directly grow a tree from the goal. To make use of the idea of growing multiple trees, the goal manifold is first randomly sampled and several goal configurations are generated. Then, random trees are grown from the initial configuration and the random goal configurations. The idea of generating several goals for manipulation planning was proposed in [Diankov 08].

Goal Manifold Sampling

A goal configuration is generated using the following algorithm:

1. Sample a random configuration $\mathbf{q}_{rand}$ in $\mathcal{CS}$ with uniform distribution.
2. Call `SolveConstraints` (Algorithm 3) on $\mathbf{q}_{rand}$, with $f(q)$ defined by the intersection of the planning and goal manifolds $\mathcal{M} \cap \mathcal{M}_g$.
3. If success, check for collisions.

Figure 2.1 shows resulting random configurations which satisfy both balance ($\mathcal{M}$) and reaching ($\mathcal{M}_g$) constraints for the HRP-2 robot.

Random Extensions on a Constrained Manifold

The RRT algorithm is described in Algorithm 1. Figure 2.2 shows an extension of the classic RRT algorithm, from a configuration already in the tree $\mathbf{q}_{near}$ towards a random configuration $\mathbf{q}_{rand}$.

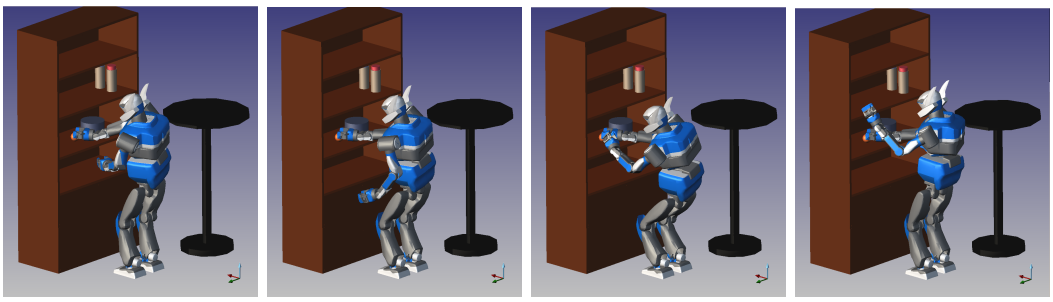


Figure 2.1: Random goal configurations solving a reaching task. All the configurations are balanced and collision-free, and the right hand of the robot reaches the orange ball.

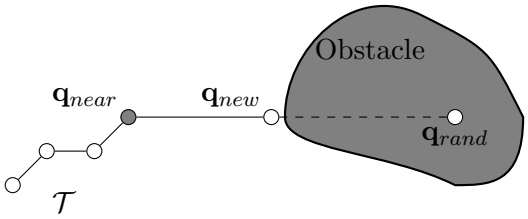


Figure 2.2: One step of extension of the RRT algorithm. The algorithm tries to add the longest possible edge from $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}$, while avoiding collisions.

The equivalent random extension on a constrained manifold $\mathcal{M}$, defined by the constraint function f, starts from a valid configuration $\mathbf{q}_{near} \in \mathcal{M}$, and extends the tree towards a random configuration $\mathbf{q}_{rand}$, while keeping the constraints satisfied. Note that $\mathbf{q}_{rand} \notin \mathcal{M}$. Extension attempts orthogonal to $\mathcal{M}$ are useless, as newly added edges have to be included in $\mathcal{M}$. To extend in directions that follow the directions of $\mathcal{M}$, we rely on Jacobian-based inverse kinematics. Algorithm 4 presents the adaptation of the classic extend function, and Figure 2.3 illustrates this extension. The idea is to first project $\mathbf{q}_{rand}$ on the tangent space to $\mathcal{M}$ at $\mathbf{q}_{near}$. Let us call the projected configuration $\mathbf{q}_{rand}{}'$. Let $\mathbf{q}_{rand}{}''$ be the result of a call to $\texttt{SolveConstraints}(\mathbf{q}_{rand}{}', \mathbf{f}, \epsilon)$. It is the projection of $\mathbf{q}_{rand}{}'$ on $\mathcal{M}$. Instead of extending the tree from $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}$, the algorithm tries to extend from $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}{}''$ while remaining on $\mathcal{M}$. While extending the tree, the configurations along the new edge are automatically projected onto $\mathcal{M}$. These projections are not very costly if the edge is close to the constrained manifold.

[Berenson 11] presents a formal proof that projection-based constrained random motion planning on a fixed dimension manifold is probabilistically complete. This proof equally applies to this algorithm.

2.1.3 Example

We present in Figure 2.4 an illustration of the use of randomized motion planning on complex manipulation problems. The humanoid robot HRP-2 faces shelves. It

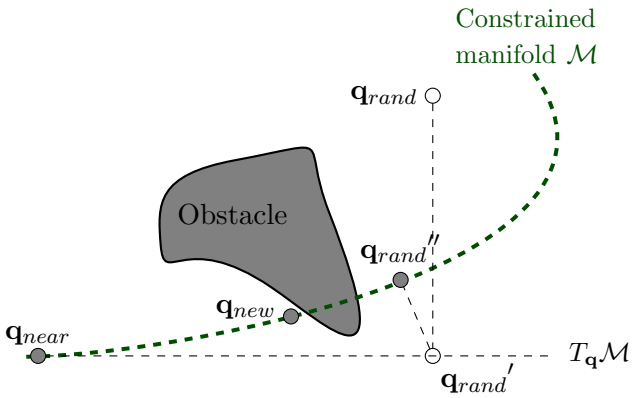


Figure 2.3: One step of constrained extension illustrating Algorithm 4: $\mathbf{q}_{rand}$ is first projected on $T_{\mathbf{q}}\mathcal{M}$ the tangent space of $\mathcal{M}$. $\mathbf{q}_{rand}{}'$ is then projected onto $\mathcal{M}$ at $\mathbf{q}_{rand}{}''$. A classic RRT extension tries to go as far as possible from $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}{}''$ while remaining on $\mathcal{M}$. $\mathbf{q}_{new}$ is then returned.

Algorithm 4 `ConstrainedExtend`($\mathcal{T}$,$\mathbf{q}_{near}$,$\mathbf{q}_{rand}$, f, ϵ)

$d \leftarrow$ `Distance`($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$)
$q \leftarrow \mathbf{q}_{near}$
while $d > \epsilon$ **do**
  $\mathbf{q}_{rand}{}' \leftarrow$ `OrthogonalProject`($\mathbf{q}_{rand}, T_q\mathcal{M}$)
  $\mathbf{q}_{rand}{}'' \leftarrow$ `SolveConstraints`($\mathbf{q}_{rand}{}', f, \epsilon$)
  $d \leftarrow$ `Distance`($\mathbf{q}$,$\mathbf{q}_{rand}{}''$)
  $\mathbf{q} \leftarrow \mathbf{q}_{rand}{}''$
end while
$\mathbf{q}_{new} \leftarrow$ RRT::Extend($\mathcal{T}$,$\mathbf{q}_{near}$,$\mathbf{q}_{rand}{}''$)

	min	max	average	average per problem
number of nodes	43.00	481.00	102.70	
goal generation time (s)	1.00	1.56	1.22	
planning time (s)	67.36	376.84	134.28	44.76

Table 2.1: Experimental results on 20 runs: Each run consists of 3 motion planning problems and 2 goal generations for the three phases. Time is expressed in seconds.

has to: (i) grasp a ball lying on a shelf, (ii) put it on a higher shelf, (iii) come back to a natural rest configuration. We can thus define three separate constrained motion planning problems where the planning manifold $\mathcal{M}$ is the static balance manifold defined in 2.1.2; the goal manifold of problem (i) is defined by a hand pose constraint (the hand must be horizontal and its position has to coincide with the ball initial position), and a gaze constraint (the robot has to look at the ball in its initial position). Similarly, the goal manifold of problem (ii) is defined by hand and gaze constraints that correspond to the position of the ball on the higher shelf. Finally, we define the rest configuration as the single goal configuration for problem (iii).

The goal configuration in phase (i) is in a narrow passage. Note that for phases (i) and (iii) the ball is also considered as an obstacle. This is necessary to prevent the robot grasping hand from colliding with the ball during the approach and retraction phase.

For the two reaching motions in (i) and (ii), we first generate 8 random goal configurations (Section 2.1.2), then we solve the three constrained motion planning problems separately. As randomized motion planning algorithms produce log paths, a classic shortcut method is used to optimize and shorten them.

This set of motion planning problems was run 20 times; results are compiled in Table 2.1. The performance of `SolveConstraints` (Algorithm 3) is also measured when used to project configurations on $\mathcal{M}$; the average number of iterations is 6.5 per call, and the success rate, i.e. the ratio of the number of successfully projected configurations over the total number of calls, is above 95 percent. This success rate, high as it is, could be further improved by sampling a better initial configuration of $\mathcal{CS}$, for example by introducing a heuristic bias towards statically balanced configurations.

2.1.4 Extension to Collision-Free Walk Planning

While the previous algorithm considers motion planning on a single submanifold $\mathcal{M}$ of $\mathcal{CS}$, similar strategies can be adopted to explore the union of submanifolds $\cup\mathcal{M}_i$ and achieve quasi-static multi-step planning for walking and free-climbing robots, see [Bretl 06, Hauser 10a]. These methods can be seen as very generic and offer the property of probabilistic completeness; they are not however directly applicable to humanoid dynamically balanced locomotion: they can only produce statically

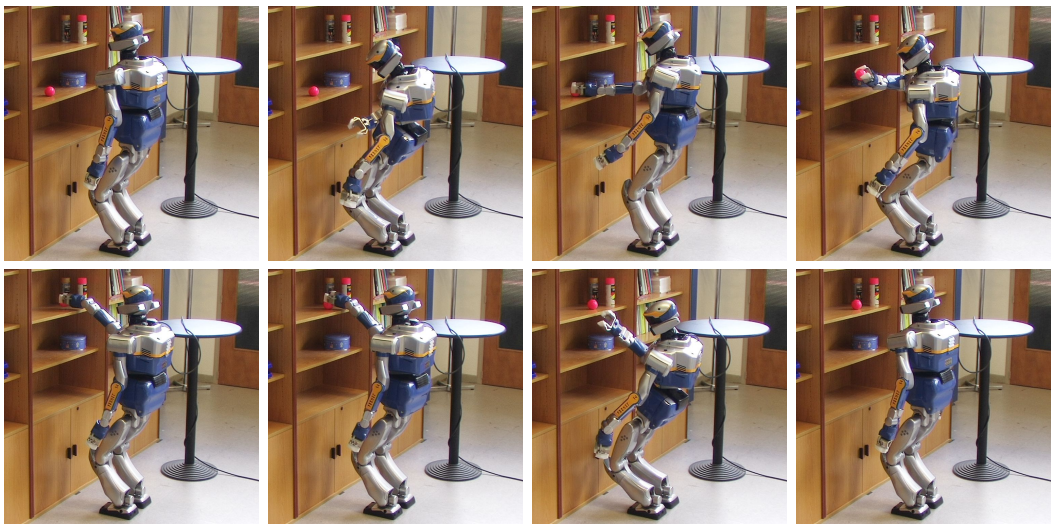


Figure 2.4: HRP-2 grabs a ball on a shelf, puts it on another shelf, and comes back to a rest position. Static balance constraints are enforced along the path, and the intermediary goals consisting in grasping and displacing the ball are defined implicitly as inverse kinematics constraints.

balanced walking paths, which restricts the scope of feasible motions. Furthermore, it is not obvious how they can be transformed to dynamically balanced motions while guaranteeing that this transformation will always succeed and not lead to unforeseen collisions.

Other recent contributions to the field of locomotion planning include algorithms considering the dynamics at the planning phase [Glassman 10, Shkolnik 11]. This leads to a growth of algorithmic complexity, particularly costly for high-dimensional systems such as humanoid robots, which can explain why such techniques have not yet been used on humanoid robotic platforms so far.

2.2 From Geometric Paths to Feasible Motions: Small-Space Controllability

Let us recalls the definition of small-space controllability and its use in motion planning.

A robotic system is controllable if for any two configurations $\mathbf{q}_1$ and $\mathbf{q}_2$, there exists a trajectory going from $\mathbf{q}_1$ to $\mathbf{q}_2$ in a finite time interval. It is *small-space controllable* if for all configurations $\mathbf{q}$, for all $\epsilon > 0$, there exists $\eta > 0$ such that all the configurations contained in the ball of center $\mathbf{q}$ and radius η are reachable, in a finite time interval, by trajectories included in the ball of center $\mathbf{q}$ and radius ϵ. Figure 2.5 shows an illustration of this property.

The main consequence of this property in motion planning is the following theorem, that shows how planning for dynamic systems is reduced to geometric planning

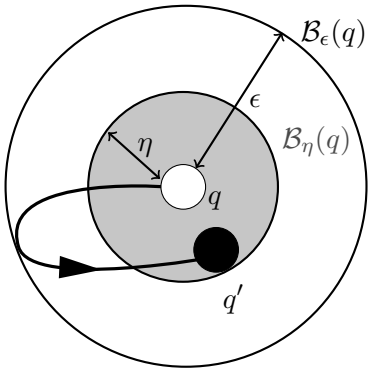


Figure 2.5: The small-space controllability local property: any configuration q' at a distance less than η is reachable from $\mathbf{q}$ by an admissible trajectory included in a ball of size ϵ.

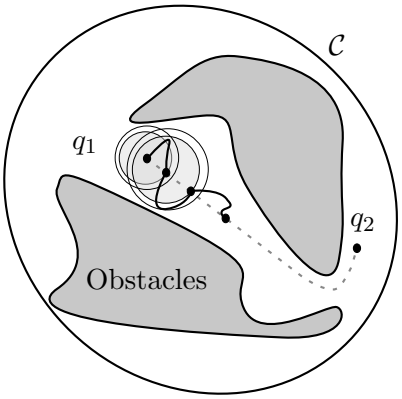


Figure 2.6: Small-space controllability in motion planning. A collision-free path from $\mathbf{q}_1$ to $\mathbf{q}_2$ is approximated by collision-free and admissible trajectories by using the local property.

thanks to the small-space controllability property:

Theorem 1. *Any collision-free path of a small-space controllable system can be approximated by a sequence of both collision-free and admissible (or feasible) trajectories. Thus, small-space controllability reduces trajectory planning problems to geometric path planning problems.*

Figure 2.6 shows an example of collision-free path approximation by admissible collision-free sub-trajectories. The convergence of this algorithm is guaranteed by the small-space controllability property.

This result has been long known and used in motion planning, in particular for non-holonomic systems. A detailed proof can be found in [Laumond 94]. We present a sketch of the proof to give an intuition about the corresponding algorithm.

Proof of Theorem 1. Let $\mathcal{CS}$ be the configuration space of a small-space controllable

robot, and $\mathcal{CS}_{free} \subset \mathcal{CS}$ the set of collision-free configurations. We consider in-contact configurations as colliding, so $\mathcal{CS}_{free}$ is an open set. Let $\tau : [0,1] \to \mathcal{CS}_{free}$ be a collision-free path. Thus for all $x \in [0,1]$, $\tau(x) \in \mathcal{CS}_{free}$, there exists ϵ_x such that the open ball $B(\tau(x), \epsilon_x)$ of center $\tau(x)$ and radius ϵ_x is included in $\mathcal{CS}_{free}$. The small-space controllability property states that for all x, there exists $\eta_x > 0$ such that every configuration $q \in B(\tau(x), \eta_x)$ is reachable from $\tau(x)$ by a trajectory included in $B(\tau(x), \epsilon_x)$.

The set of open balls $(B(\tau(x), \eta_x))_{x\in[0,1]}$ forms an open cover of $\tau([0,1])$ which is compact. The Heine-Borel theorem [Fitzpatrick 06] states that there exists a finite subcover $(B(\tau(x_i), \eta_{x_i}))_{i\in\{1,\dots,n\}}$ of $\tau([0,1])$. To this finite subcover corresponds a finite number of feasible trajectories, going from $\tau(0)$ to $\tau(1)$, included in the union of $(B(\tau(x_i), \epsilon_{x_i}))_{i\in\{1,\dots,n\}}$, and thus in $\mathcal{CS}_{free}$. This concludes the proof. □

Small-Time *versus* Small-Space Controllability

In the control theory literature, the property used is usually *small-time controllability*, which states that for all configurations $\mathbf{q}$, for all times $T > 0$, the set of configurations accessible from $\mathbf{q}$ in time less than T forms a neighborhood of $\mathbf{q}$. When accelerations and velocities are bounded, small-time controllability implies small-space controllability. This is why a lot of motion planning previous work only refers to the sufficient small-time controllability property. However, the converse is not necessarily true: a system can be small-space controllable and not small-time, if the trajectories generated by its controller are arbitrarily long. The important property, regarding motion planning application, is small-space controllability, as Theorem 1 shows. In the following, we show that legged robots are small-space controllable, but not that they are small-time controllable. In fact, the control method that we present does not follow the small-time controllability property. For the sake of clarity, we have chosen to make the distinction between these two controllability properties.

2.3 Contribution

The main contribution of this chapter is a whole-body motion planner for humanoid robots that computes collision-free walking trajectories, based on exact models of both the robot and its environment. It is used to solve manipulation tasks that may require walking. The first stage of our algorithm uses a sampling-based constrained motion planner and computes a collision-free statically balanced path for a robot which can be fixed or sliding on the ground.

Another contribution of this chapter is the formal proof that dynamic walking makes humanoid robots small-space controllable, with the direct implication that this first path can always be approximated by a dynamically balanced, collision-free walking trajectory. We have implemented this well-grounded method, and the results have been validated on the HRP-2 robot.

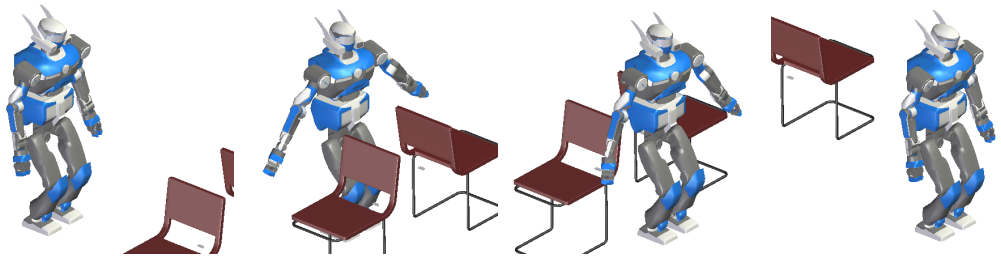


Figure 2.7: Collision-free statically balanced path for a humanoid robot sliding on the ground.

Section 2.4 generalizes the constrained motion planning algorithm to problems that require locomotion. The generalization is well-grounded, and based on a controllability property of legged robots demonstrated in the chapter. Section 2.5 presents some experimental results, and Section 2.6 discusses the limitations and potential future work of our method.

2.4 From Statically Balanced Paths to Dynamic Walk Trajectories

The previous section has presented a simple algorithm that solves manipulation planning problems on a given constraint manifold $\mathcal{M}$ of $\mathcal{CS}$. If we use this algorithm with static balance constraints without fixing globally the robot foot positions, it generates statically balanced paths for a robot *sliding* on the ground. Fig 2.7 shows an example of a whole-body collision-free path for a robot passing between two chairs. Since in reality a legged robot cannot slide on a regular floor, such paths are physically unfeasible. They are, however, easier to generate than feasible dynamic walking trajectories because only geometric constraints are considered at planning time.

This section presents a *constructive* proof that any such statically balanced, collision-free path for a legged robot sliding on the ground can be approximated by a dynamically balanced, collision-free walk trajectory. The proof is based on ideas from control theory, in particular small-space controllability. It also uses the fact that balance criteria for dynamic walking are different from the ones for static balance.

Section 2.4.1 proves that a dynamically walking legged robot is small-space controllable, while a quasi-statically walking legged robot is not. Section 2.4.2 shows how this property is used to approximate collision-free statically balanced paths by dynamic walking trajectories.

2.4.1 Small-Space Controllability of Dynamically Walking Robots

This section discusses a walking robot small-space controllability. To clarify the presentation, we consider a simplified model of a legged robot consisting of two feet of zero mass and a point mass free to move in three dimensions. We do not consider the kinematic chains between the feet and the mass. The robot is walking on a flat terrain, and the feet are assumed to have a positive surface. For our presentation, it is not necessary to consider the foot height, so the configuration space of the robot is:

$$\mathcal{CS} = SE(2) \times SE(2) \times \mathbb{R}^3 \tag{2.1}$$

It is of dimension 9.

The balanced walking conditions for a quasi-static walking robot are that the point mass, or CoM, should always be over the support polygon (the convex hull of the two feet), and one foot can move iff the CoM is over the other foot. Similarly, the walking conditions for a dynamic walking robot are that the ZMP should always be in the robot support polygon, and one foot can move iff the ZMP is over the other foot. Under these assumptions, the following result holds:

Theorem 2. *A quasi-statically walking robot is not small-space controllable. A dynamically walking robot is.*

Proof of Theorem 2. The first claim is straightforward. Let the robot be in a configuration $\mathbf{q}$ where the two feet are separated by a positive distance. Let $L > 0$ be the positive horizontal distance between the CoM and the left foot (if the CoM is over the left foot, we can consider similarly the right foot). For all $\epsilon < L$, any valid trajectory starting from $\mathbf{q}$, included in the ball of center $\mathbf{q}$ and radius ϵ $B(\mathbf{q}, \epsilon)$, is such that the CoM is never over the left foot. Given the quasi-static walking conditions, the right foot of the robot is fixed along the trajectory. Thus, the set of accessible configurations from $\mathbf{q}$ by staying inside $B(\mathbf{q}, \epsilon)$ does not form a neighborhood of $\mathbf{q}$, since it does not include any configuration where the right foot has moved. This shows that the robot is not small-space controllable.

Let us now consider a dynamically walking robot. If the CoM is not over the edge of the support polygon, it is possible to move it in a quasi-static way inside a neighborhood of its current position that projects itself over the support polygon. It is thus sufficient and necessary to prove that for all $\epsilon > 0$, it is possible to move the feet while keeping the CoM inside a neighborhood of size ϵ. Let such $\epsilon > 0$ be arbitrarily fixed.

Let us recall the equations giving the ZMP horizontal coordinates (p_x, p_y) as functions of CoM horizontal coordinates (x, y) in the cart-table model, as presented in Section 1.3.2:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x - \frac{z_c}{g}\ddot{x} \\ y - \frac{z_c}{g}\ddot{y} \end{pmatrix} \tag{2.2}$$

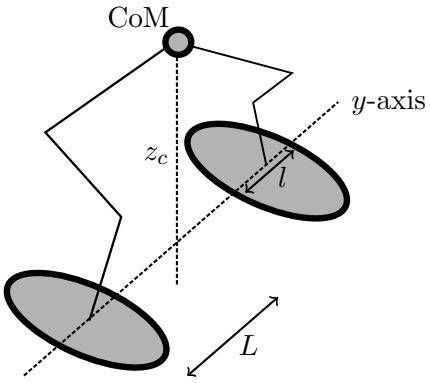


Figure 2.8: Simplified model of a legged robot. The CoM is at $(0, 0, z_c)$, the two feet are flat on the ground, aligned with the y-axis, at a horizontal distance L from the CoM.

where z_c is the constant height of the CoM and g is the gravity constant. In the following we will note $\omega_0 = \sqrt{\frac{g}{z_c}}$.

Without loss of generality, let us assume that the robot is in a configuration in which the CoM is at the horizontal position $(0, 0)$, the foot centers are aligned with the y-axis and the horizontal distance between the CoM and either of the foot centers is L. To achieve dynamically balanced walking, we aim at making $p_y(t)$ oscillate between $-L$ and L. To move the ZMP under a given foot, only the y coordinate of the CoM is of interest. Thus, we will keep the x coordinates of the CoM and ZMP constant equal to 0. By assumption, the feet have a positive surface, let $l > 0$ be such that the length of the section of a foot along the y-axis is greater than l. Figure 2.8 summarizes the notations used in the following.

The idea of this proof is to use the form of Equation (2.2) to apply a scaling factor between the amplitude of the oscillations of the CoM and of the ZMP. The faster the CoM oscillates, the bigger is the amplitude of the ZMP oscillations. Following is a formalization of this idea.

For $\omega > 0$, assuming the CoM follows the trajectory $y(t) = \epsilon \sin(\omega t)$, Equation (2.2) gives:

$$p_y(t) = (1 + \left(\frac{\omega}{\omega_0}\right)^2)\epsilon \sin(\omega t) \tag{2.3}$$

The amplitude of the oscillations of y is multiplied by a factor $(1 + \left(\frac{\omega}{\omega_0}\right)^2)$. Choosing $\omega = \omega_0\sqrt{\frac{L}{\epsilon} - 1}$ makes p_y oscillate between $-L$ and L, while y oscillates between $-\epsilon$ and ϵ. At time $t_l^{(n)} = n\frac{2\pi}{\omega} + \frac{\pi/2}{\omega}$, the ZMP is located at the center of the left foot, the robot can move its right foot and at time $t_r^{(n)} = n\frac{2\pi}{\omega} + \frac{3\pi/2}{\omega}$ the ZMP is located at the center of the right foot, the robot can move its left foot.

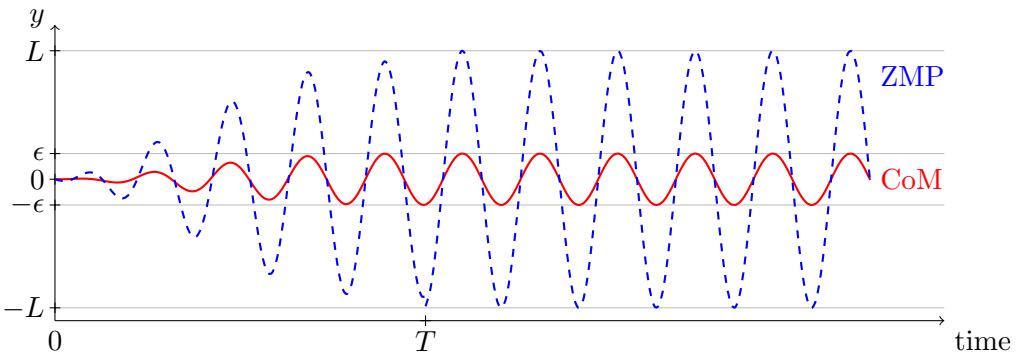


Figure 2.9: CoM motion (solid line) along y axis. The CoM stays in the interval $[-\epsilon, \epsilon]$ while during steady state ($t \geq T$), the ZMP (dashed line) oscillates between the centers of the feet, which allows in-place walk.

Starting from a static configuration at time ($t = 0$), we cannot apply directly a control $y(t) = \epsilon \sin(\omega t)$ because it generates a discontinuity in the velocity of the CoM at time ($t = 0$). To overcome this discontinuity, we go through a transient state between ($t = 0$) and ($t = T$) for some $T > 0$. Let $f : [0, T] \to [0, 1]$ be an increasing function of class C^∞ such that $f(0) = 0$, $\dot{f}(0) = 0$, $f(T) = 1$, $\dot{f}(T) = 0$ and $\ddot{f}(T) = 0$. We can explicitly construct such an f with a spline of degree 4. We also request that for all $t \in [0, T]$, $|2\epsilon\dot{f}(t)\frac{\omega}{\omega_0^2}| \leq \frac{l}{4}$ and $|\epsilon\ddot{f}(t)/\omega_0^2| \leq \frac{l}{4}$. These inequalities will be used to bound the trajectory of the ZMP. We can guarantee them by choosing T large enough. Let us now consider the following CoM motion:

$$y(t) = \begin{cases} f(t)\epsilon \sin(\omega t) & \text{if } t \in [0, T] \\ \epsilon \sin(\omega t) & \text{if } t \geq T \end{cases} \tag{2.4}$$

One can check that y is of class C^2 over $\mathbb{R}_+$, and that $\dot{f}(0) = 0$. When $t \geq T$, the robot is in the steady state described above and can successively move either of its feet inside small neighborhoods. The last point to check is that for $t \in [0, T]$ $p_y(t)$ stays inside the support polygon of the robot. The calculation of the successive derivatives of y gives:

$$\begin{aligned} p_y(t) = \quad & f(t)\epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2) \sin(\omega t) \\ & + 2\epsilon\dot{f}(t)\frac{\omega}{\omega_0^2}\cos(\omega t) \\ & + \frac{\epsilon}{\omega_0^2}\ddot{f}(t)\sin(\omega t) \end{aligned} \tag{2.5}$$

For all $t \in [0, T]$, $f(t)\epsilon(1 + \frac{\omega}{\omega_0}^2) \sin(\omega t)$ lies between $-L$ and L. The bounds on the derivatives of f guarantee that $p_y(t)$ lies between $-L - l/2$ and $L + l/2$, which means that the ZMP stays inside the support polygon. Figure 2.9 shows an

example of CoM motion on the y axis and the corresponding ZMP motion. Once in steady in-place walking state, the robot can come back to a static state by applying a symmetric transient state used to decrease gradually the amplitude of the oscillations of the CoM without generating a discontinuity in the first derivative of the control.

We have thus exhibited a continuous control scheme that allows to move any of the feet in any direction, while keeping the CoM inside an arbitrarily small neighborhood. This concludes the proof. □

Remarks

Generalization to a complete model: We have not extended the above proof to any legged robot model since empirically, the table cart model describes a large part of the dynamics of a walking humanoid robot. Although of little practical interest, the generalization of the proof does not seem very difficult to achieve. As an insight, the difference between the table cart model and the full size humanoid robot is due to the derivative of the angular momentum and to the vertical acceleration of the center of mass. These perturbations can be made as small as desired along the sliding path by following the sliding path as slowly as necessary. The derivatives of the angular momentum produced by the stepping motion can also be made as small as desired by making the step height as small as necessary and by using recent results on properties of joint trajectories induced by end-effector motions [Zanchettin 12].

Use of ZMP preview controller: The control strategy presented in the above proof may generate very long trajectories, because of the transient states at the beginning and end of the locomotion. In the actual implementation, we have chosen to generate CoM motions with a ZMP preview controller, as presented in [Kajita 03]. We have observed experimentally that the amplitude of CoM trajectories decreases when the frequency of steps increases. Our current ZMP preview controller relies on the cart-table model approximation. To make this approximation valid, we fix the height of the robot CoM during walk, as well as the vertical orientation of the robot waist. These geometric constraints are also applied when planning statically balanced paths, to ensure that the paths can be approximated by dynamic walk trajectories. Note that this is due to our current ZMP preview controller implementation, and does not affect the generality of the small-space controllability result presented above.

Relying on the cart-table model approximation means that the angular momentum induced by arm movements for instance can lead to non dynamically balanced walking motion. We thus implement the ZMP filtering stage proposed in [Kajita 03] to compute the exact ZMP, take into account the full dynamics of the robot and generate feasible trajectories.

Velocity of CoM: The theoretical result presented in this section implies that any collision-free path can be approximated by a sequence of admissible and collision-

free trajectories. However, the theorem depends on a control law that generates trajectories with unbounded velocities for the CoM, when the input path is close to obstacles. The humanoid robot hardware (actuators, mechanical structure, etc.) may be a limitation to such trajectories. To prevent the generated CoM oscillations from being too fast, one has to require that the statically balanced path is included inside an ϵ-radius tube of the free space, where ϵ depends on the physical capabilities of the robot.

2.4.2 Application: Dynamic Approximation of a Statically Balanced Sliding Path

The algorithm that animates a statically balanced path into a dynamically balanced walk trajectory has been inspired by the previous small-space controllability proof. Given a statically balanced path P verifying the cart-table model approximation constraints, we start by placing footprints corresponding to the nominal walk pattern of the robot. Given the footprints, we compute a ZMP trajectory, derive foot trajectories, and a preview controller returns the corresponding CoM trajectory. Classic numerical Jacobian-based prioritized inverse kinematics methods prove to be very useful to generate a dynamic walking trajectory while trying to accomplish secondary tasks, such as following a reference configuration trajectory. We use the framework called Generalized inverse kinematics (Gik) developed in [Yoshida 06].

The hierarchy of tasks (referred to as *GikTasks* in Algorithm 5) applied to the robot to generate a dynamic walking motion is – in decreasing priority order:

1. Positions and orientations of feet,
2. Horizontal position of the CoM,
3. Height of the CoM,
4. Verticality of the waist,
5. Configuration task towards corresponding configuration in P.

Tasks (1) and (2) generate a dynamically balanced motion by using the simplified cart-table model and the ZMP criterion. Tasks (3) and (4) ensure that the resulting motion is well described by the cart-table model. Task (5) is used to approximate P as well as possible given the walk parameters.

Because it comes at the lowest priority, task (5) is not necessarily fulfilled in the resulting trajectory. Hence, collisions may appear when animating P, if the resulting trajectory diverges too much from the initial sliding path. If so, it is necessary to approximate more closely P by a walk trajectory. To do so, we use the small-space controllability property of the system shown in the previous section. The way we use this property is inspired by similar results in non-holonomic mobile robot control presented in [Laumond 94].

If the animated trajectory collides with the environment, we cut the initial path P into two sub-paths, that we try to animate recursively. When the paths to animate are too short for the robot nominal walk parameters, we accelerate the steps, and decrease the maximum height of the moving foot. As shown in previous section, the walk trajectory corresponding to smaller and faster steps converges toward the sliding path. Algorithm 5 shows pseudo-code that takes a sliding path P as input and returns a collision-free walk trajectory. Figure 2.10 shows a sketch of the method.

Algorithm 5 `FindDynamicTrajectory`(Path P)

$Footprints \leftarrow$ ComputeFootprints(P)
$GikTasks$.addFootprintTask($Footprints$)
$GikTasks$.addWaistTask()
$GikTasks$.addConfigurationTask(P)
$DynamicTrajectory \leftarrow$ ComputeWalkTrajectory($GikTasks$)
if (CheckForCollisions($DynamicTrajectory$) = Colliding) **then**
  $(p_1, p_2) \leftarrow$ CutInHalf(p)
  $DT_1 \leftarrow$ FindDynamicTrajectory(p_1)
  $DT_2 \leftarrow$ FindDynamicTrajectory(p_2)
  return Concatenate(DT_1, DT_2)
else
  return $DynamicTrajectory$
end if

2.5 Experimental Results

The motion planning algorithms presented in this chapter have been implemented using KineoWorks^{TM}[Laumond 06]. The planning times have been measured on an Intel Core 2 Duo 2.13 GHz PC with 2 GB of RAM. Evaluation of the randomized algorithm has been conducted by executing 500 trials on each scenario using two flavors of RRT: the classic RRT and IPP-RRT [Ferre 04]. We present the results in Figures A.1, A.2 and A.3 in Appendix A.

Our whole-body motion planner generates a robot configuration trajectory that is sampled at a 200 Hz rate and stored in a file. This file can then be used to play the trajectory in open-loop on the HRP-2 robot, which is position-controlled. Scenarios in Sections 2.5.1 and 2.5.3 were both successfully executed.

In this work, to get "nicer" walking motions in the experiments, we require the foot positions to be fixed with respect to each other, and the CoM to be projected in the center of the support polygon during the sliding path planning stage.

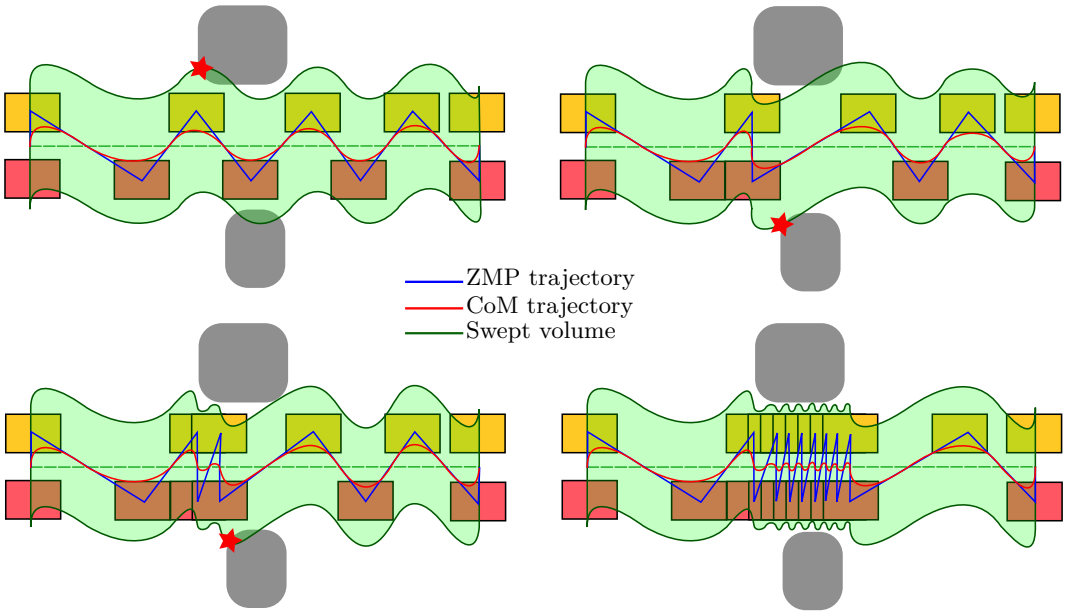


Figure 2.10: The first three steps of algorithm 5 are shown (top left and right, bottom left). Starting from a sliding collision-free path (dashed), a walking trajectory is computed using nominal walk parameters. If collisions are detected, the path is recursively cut and animated until a collision-free dynamic walking trajectory is found (bottom right). Notice that shorter and faster steps in the middle of the trajectory lead to smaller deviations of the CoM trajectory (in red) from the original path.

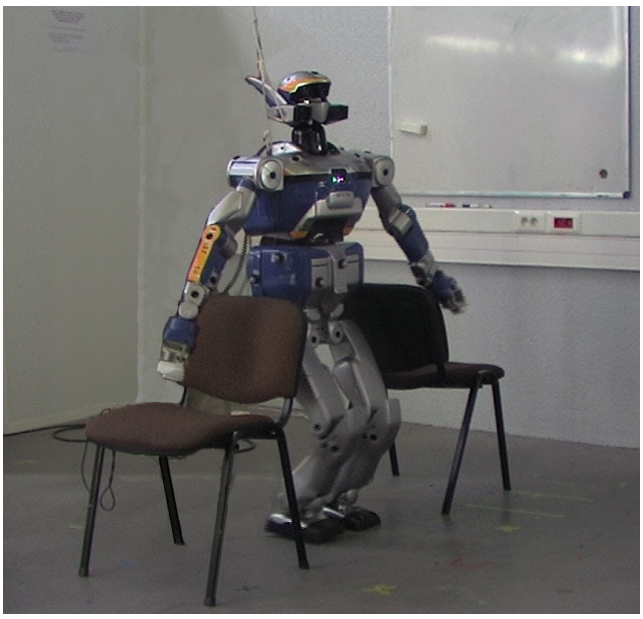


Figure 2.11: The robot HRP-2 passing between two chairs. In this kind of environment whole-body collision avoidance is needed during locomotion.

2.5.1 Passing between two chairs

The environment shown in Figure 2.11 and 2.7 was presented in Section 1.7.1. There, the motion planning problem is solved with a bounding box method, leading the robot to walk sideways between the two chairs. The method presented in this chapter generates a locomotion trajectory in which the robot walks forward, which may be required if the robot has to use vision during locomotion. The first planning stage requires 1 s on average. The animation of the sliding path presented in Figure 2.7 uses 66.5 s of computation time.

Figure 2.12 shows the horizontal trajectory of the robot CoM during locomotion. The amplitude of the oscillations decreases when passing between the chairs. This motion has been validated on a real HRP-2 platform.

2.5.2 Walking among floating obstacles

In the environment shown in Figure 2.13, the robot has to find a way among floating obstacles. In this environment neither bounding box nor footstep planning strategies could find a collision-free walk trajectory. The first planning stage requires 53 s on average, and the animation of the trajectory presented in Figure 2.13 uses 339.5 s of computation time. Figure 2.14 shows the robot CoM trajectory during locomotion.

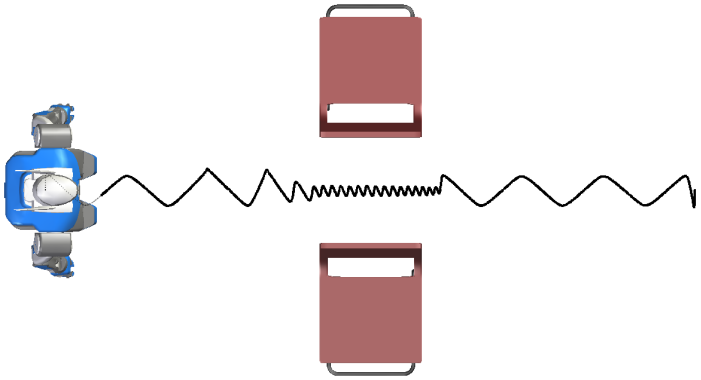


Figure 2.12: Horizontal trajectory of the robot CoM during locomotion. When the robot is close to obstacles, the amplitude of the oscillations decreases.

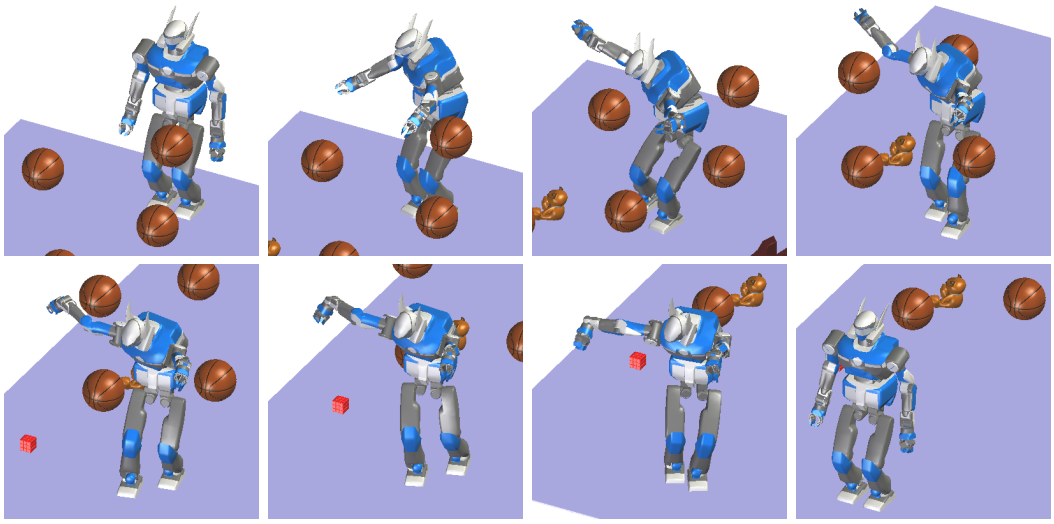


Figure 2.13: Solution path for a cluttered environment, the robot walks among floating obstacles.

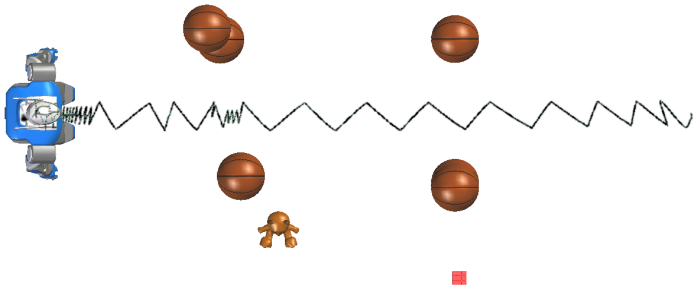


Figure 2.14: Horizontal trajectory of the robot CoM during locomotion.

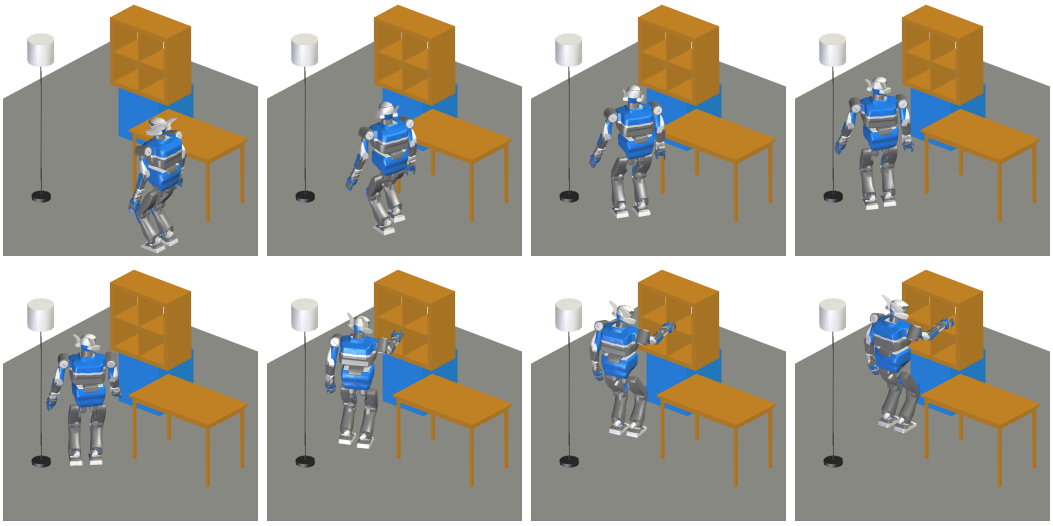


Figure 2.15: Solution path for a hand reaching problem in an apartment. The goal is implicitly defined as an inverse kinematics task.

2.5.3 'Put the ball on a shelf'

In the problem shown in Figure 2.15 the robot has to put a ball on a shelf, in a constrained apartment environment. The final configuration is defined implicitly as a desired hand position. We have generated automatically goal configurations solving the task, as described in Section 2.1.2 . Then, we have applied our planner to generate a whole-body walking motion that solves the hand reaching task.

The solution sliding path is constrained between the table on the right and the lamp on the left. This passage is too narrow for the robot nominal walk parameters. When executing the walk motion resulting from our algorithm, the robot left hand is only a few centimeters away from the lamp.

The first planning stage requires 15 s on average, and the animation of the resulting walk motion presented in Figure 2.15 requires around 190 s of computation time. Figure 2.16 shows the robot CoM trajectory during locomotion. Figure 2.17 shows some snapshots taken from the motion execution on the real robot HRP-2.

2.6 Discussion and Future Work

This section lists some limitations of the current methods, and discusses potential future work to overcome them.

Stepping over obstacles Because of the kinematic constraints we apply at the planning stage, we are not able yet to plan motions where the robot steps over obstacles, while this is an important feature of humanoid robots. Nevertheless, because we compute collision queries on an exact model of the robot, our method is

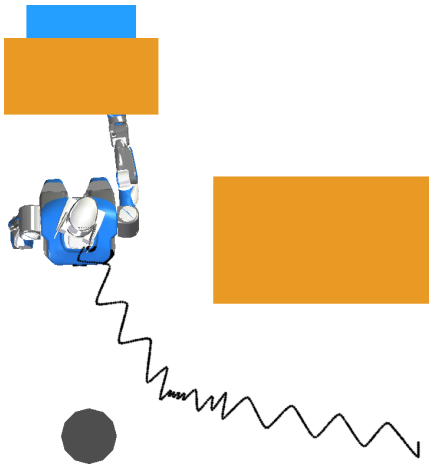


Figure 2.16: Horizontal trajectory of the robot CoM during locomotion.

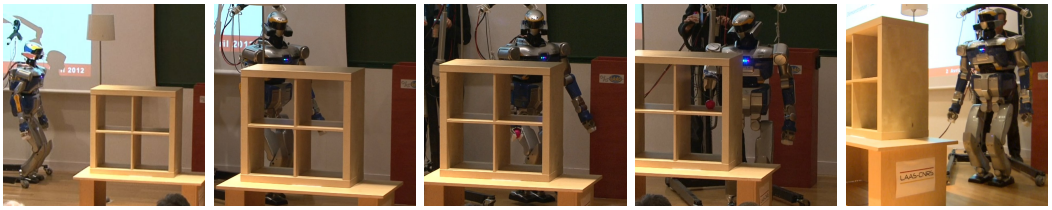


Figure 2.17: Execution of the walking trajectory by HRP-2 on stage. The robot first goes to the shelves to release the ball, then comes back to a rest position.

able to generate paths where obstacles pass between the feet of the robot. One way of dealing with this issue would be to incorporate the continuous footstep planner described in [Perrin 11] in the planning phase; it would allow stepping over obstacles, while whole-body collision-avoidance would be solved by the algorithm presented in this chapter. The main challenge with this approach would be then to verify that the small-space controllability property still holds.

Environment representation The experimental setup assumes perfect knowledge of the environment. This can be guaranteed during experiments by using calibrated objects and motion capture systems. This indeed allows us to focus on complex motion planning problems. The perception problem, interesting as it is, is thus completely decoupled from the planning problem in this work. Experiments in [Nakhaei 08, Dang 12] have shown that it is possible to build a representation of the environment using stereo vision, and use it to plan collision-free motions for humanoid robots.

Trajectory following The setup also assumes perfect execution of the plan. It can be critical here, since non-nominal stepping may cause the robot to drift away from the planned trajectory, and collide with obstacles. It is thus necessary to devise a localization-based controller which, based on the current state of the robot, will locally modify the robot future states to make sure the planned trajectory is followed, see [Moulard 12b]. This approach works only if local perturbations are applied to the system or environment between the planning and execution phases. One way to guarantee a proper motion execution would be to establish a planning-control loop; under the assumption that each component is fast enough, this loop could take into account all modifications in the environment or the robot global position by continuously re-planning new trajectories, see [Baudouin 11]. Note that whatever approach is chosen, accurate localization methods must be used, and multiple sensor sources might need to be fused to obtain relevant information about the humanoid robot pose. An example of such methods can be found in [Hornung 10], where accurate localization information is obtained from multiple sensors in complex environments including staircases.

2.7 Conclusion

In this chapter, we have used a simple algorithm for constrained motion planning within a novel, well-grounded strategy for humanoid whole-body manipulation planning including locomotion. The locomotion algorithm is based on a formal small-space controllability property of humanoid robots. An important point is that this strategy only holds for dynamic walking robots, and not for quasi-static walking ones. We have used our motion planner on different challenging examples, and validated the generated motions on a real platform. We have discussed the limits and potential extensions of our method, and we plan to address them in future work.

This motion planning algorithm helps us achieve global planning in complex environments, but it still cannot provide us with the "best" possible trajectories with respect to a given cost function. In the next chapter, we focus on planning dynamic optimal motions for humanoid robots in complex environments, such that the generated trajectories solve both motion planning and optimal control problems. We will therefore use the same constrained motion planning algorithms which we presented in this chapter and combine them with optimal control techniques.

Chapter 3

Optimal Motion Planning for Humanoid Robots

The generation of the best possible trajectory that does not violate any constraints imposed by the environment is an ubiquitous task in both industrial and humanoid robotics. Numerous examples of successful robotic applications in the domains of motion planning and optimal control can be encountered in literature and industry. Very few however consider the more general problem of optimal motion planning for complex robots evolving in complex environments.

There are two established but still quite separate research areas that both address a part of the optimal motion planning problem, namely path planning and optimal control. This chapter aims at combining state-of-the-art developments of path planning and optimal control and to create the algorithmic foundations to tackle optimal control problems in cluttered environments. We thus propose a two-stage framework for optimal motion planning on complex robots, where a quasi-statically feasible path is first planned then optimized in order to produce a dynamically feasible trajectory. We additionally describe a simple method to automatically generate minimum bounding capsules around exact robot body geometries represented by meshes; the capsules are used to implement distance constraints for an optimal control problem solver and achieve (self-)collision avoidance. The whole framework is successfully applied to generate optimal collision-free trajectories on the humanoid robot HRP-2.

3.1 Path Planning

Sampling-based algorithms, such as Rapidly-exploring Random Trees (RRT) which were presented in Section 1.1.2, are particularly powerful when it comes to solving path planning problems in high-dimension $\mathcal{CS}$ and cluttered environments. In this chapter, we rely on the same constrained RRT algorithm which was described in Chapter 2.1.2 in order to generate statically balanced paths on a submanifold of $\mathcal{CS}$.

Let us recall that an important feature of sampling-based algorithms is their probabilistic completeness, i.e. their capacity to avoid falling into local minima and to find a solution path if it exists. They present however three drawbacks. First, due to their random sampling nature, the configuration $\mathbf{q}$ might move in a random fashion along the path P, which could lead to unnecessarily long paths. Second, we still need to apply a time parametrization in order to transform the path into a trajectory. This is a non-trivial task in the particular case of a humanoid robot, as we must ensure its dynamic balance along the trajectory. Third, the resulting paths are continuous but not C^1; to enforce this constraint, the time-parameterized motion would need to stop at each waypoint, or would leave the planned path around waypoints. Additional processing is thus needed to provide a reshaped collision-free trajectory that can be executed on the robot.

3.2 Numerical Optimization

We give in Appendix B an overview of the most successful numerical optimization techniques that can be found in the literature. We focus on Jacobian-based methods, i.e. methods that use information given by the *variations* of the function we want to minimize to find its minimizer.

3.3 Optimal Control

While the previous section described numerical optimization and its associated techniques, this section discusses the particular application of finding, for a dynamic model such as an anthropomorphic system, a trajectory that allows the model to move over the course of time from an initial state to a final state, while minimizing a certain criterion. This particular field is known as *optimal control*, and has been a major field of interest in the Control Theory and Robotics communities ever since their beginning.

Given a dynamic model, let:
- t denotes the time variable,
- $\mathbf{x}$ denotes the state vector,
- $\mathbf{u}$ denotes the control vector,
- T denotes the trajectory duration,
- L denotes the Lagrangian term (also cost rate) of the objective function,
- Φ denotes the Mayer term (also terminal cost) of the objective function,
- $\mathbf{f}$ denotes the *Ordinary Differential Equation (ODE)* of the model,
- $\mathbf{g}$ denotes the equality constraint vector function,
- $\mathbf{h}$ denotes the inequality constraint vector function,
- $\mathbf{r}$ denotes the boundary conditions vector function.

An *optimal control problem (OCP)* can be written as follows:

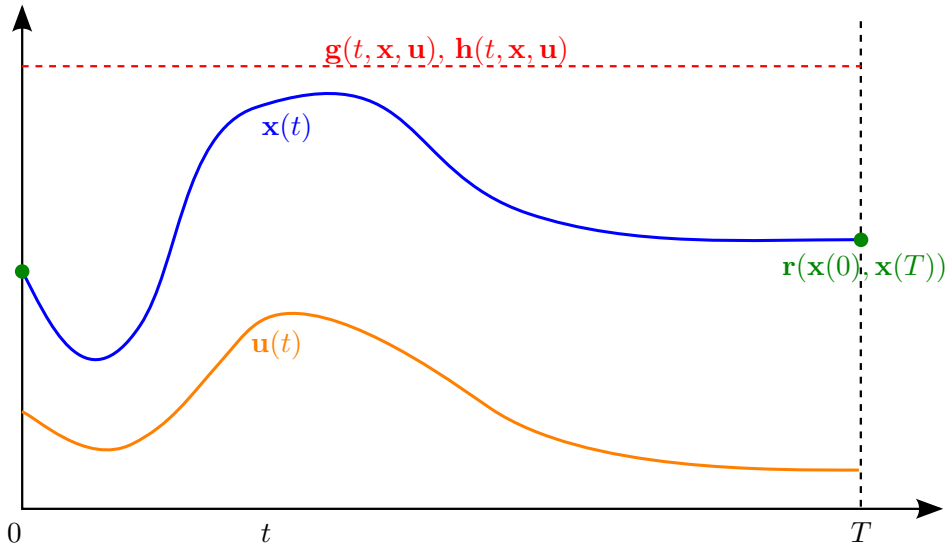


Figure 3.1: Illustration of the optimal control problem, showing the control and state vectors, path and boundary constraints. As the space of continuous functions is infinite-dimensional, the general OCP is also infinite-dimensional.

$$\min_{\mathbf{x}(\,\cdot\,),\mathbf{u}(\,\cdot\,),T} \quad J(\mathbf{x}(\,\cdot\,),\mathbf{u}(\,\cdot\,),T) = \int_0^T L(\mathbf{x}(t),\mathbf{u}(t))dt + \Phi(\mathbf{x}(T)) \tag{3.1}$$

subject to:

$$\begin{array}{rcll} \dot{\mathbf{x}}(t) & = & \mathbf{f}(t,\mathbf{x}(t),\mathbf{u}(t)), & t \in [0,T], \\ \mathbf{g}(t,\mathbf{x}(t),\mathbf{u}(t)) & = & \mathbf{0}, & t \in [0,T], \\ \mathbf{h}(t,\mathbf{x}(t),\mathbf{u}(t)) & \geq & \mathbf{0}, & t \in [0,T], \\ \mathbf{r}(\mathbf{x}(0),\mathbf{x}(T)) & = & \mathbf{0}. & \end{array} \tag{3.2}$$

Figure 3.1 summarizes the OCP. Note that it cannot be written (yet) as a finite-dimensional NLP and use associated solving methods, as $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are infinite-dimensional.

A significant number of methods which can solve the OCP have been developed in the past sixty years. They can be classified into three broad categories: dynamic programming, inverse methods and direct methods. For more details on dynamic programming and indirect methods, the interested reader can refer to [Laumond 98, Todorov 06]. We give a broad description of the first two methods and discuss direct methods more deeply, as they are heavily used in Robotics and Computer Graphics nowadays. Note that the list we establish is largely inspired by [Diehl 06]; an even more exhaustive survey can be found in [Betts 98, Betts 10].

3.3.1 Dynamic Programming

Dynamic Programming is based on Bellman's *Optimality Principle* [Bellman 65], which states that for any OCP going from an initial to a final state, we have a solution optimal control if we have a solution optimal control for the sub-OCP starting from a state reached from the initial state and going to the final state.

Let $v(\mathbf{x},t) = \int_t^T L(\mathbf{x}(t),\mathbf{u}(t))dt + \Phi(\mathbf{x}(T))$. Using this principle, the *Hamilton-Jacobi-Bellman (HJB) Partial Differential Equation (PDE)* can be derived for continuous-time systems:

$$\dot{v}(\mathbf{x},t) + \min_{\mathbf{u}(\cdot)} \Big(f(\mathbf{x},\mathbf{u})^\top \nabla_{\mathbf{x}} v(\mathbf{x},t) + L(\mathbf{x},\mathbf{u},t) \Big) = 0, \tag{3.3}$$

subject to the terminal condition:

$$v(\mathbf{x},T) = \Phi(\mathbf{x}(T)). \tag{3.4}$$

v is also called the *value function*. Equation (3.3) is the PDE which gives the necessary conditions for finding an optimal control policy. It can be solved backwards in time, just as in discrete-time dynamic programming, starting from $t = T$ and ending at $t = 0$. State-of-the-art solvers which rely on finite-element methods are then used to solve it. However, as the PDE contains partial derivatives with respect to both time and state, solvers suffer from the curse of dimensionality and the HJB equation is not used for large-scale systems in practice.

3.3.2 Indirect Methods

Another fundamental idea in optimal control is the *Maximum (or Minimum) Principle*, introduced by Pontryagin [Boltyanskii 60]. It also gives necessary conditions for optimal control, and leads to the same solutions as the optimality principle. It can be derived indirectly from the HJB equation, by first hiding the value function gradient in a *costate* vector:

$$\mathbf{p}(t) = \nabla_{\mathbf{x}} v(\mathbf{x},t), \tag{3.5}$$

and defining the *Hamiltonian* as the objective function in the HJB PDE:

$$\mathcal{H}(\mathbf{x},\mathbf{u},\mathbf{p},t) = f(\mathbf{x},\mathbf{u})^\top \mathbf{p}(t) + L(\mathbf{x},\mathbf{u},t) \tag{3.6}$$

These new definitions lead to Pontryagin's minimum principle necessary conditions:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \frac{\partial}{\partial \mathbf{p}} \mathcal{H}(\mathbf{x},\mathbf{u},\mathbf{p},t) \\ -\dot{\mathbf{p}}(t) &= \frac{\partial}{\partial \mathbf{x}} \mathcal{H}(\mathbf{x},\mathbf{u},\mathbf{p},t) \\ \mathbf{u}(t) &= \arg\min_{\mathbf{u}(\cdot)} \mathcal{H}(\mathbf{x},\mathbf{u},\mathbf{p},t). \end{aligned} \tag{3.7}$$

Remarkably, this transformation turns the HJB PDE into a $2n$-dimensional ODE which can be solved with standard boundary value problems with linear complexity. Furthermore, deriving the optimal control policy consists in minimizing the Hamiltonian, which is very easy in the particular case where controls appear linearly in the dynamics and quadratically in the cost rate.

3.3.3 Direct Methods

Dynamic programming and indirect methods are used to solve the exact OCP. Direct methods, on the other hand, rely on first discretizing the controls and/or the states: this effectively transcribes the infinite-dimensional OCP into a finite dimensional NLP, which can be solved using standard numerical optimization techniques. This allows handling all constraints more easily, and, surprisingly, can lead to better performance than exact methods, as specialized solvers can take advantage of the high sparsity of the NLP, i.e. the fact that the associated gradients contain a large number of zeros. The major drawback of such methods comes from the fact that we only obtain an approximate solution of the OCP, but a proper choice of discretization gives good results in practice. Nowadays direct method are the most commonly used methods due to their easy applicability to large-scale problems and their robustness.

Direct Single-Shooting

In *direct single-shooting methods* [Hicks 71, Sargent 78], the control vector $\mathbf{u}(t)$ is discretized on a fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$. The state vector $\mathbf{x}(t)$ is regarded as a dependent variable on $[0, T]$; numerical integration is then used in order to obtain, starting from an initial state $\mathbf{x}(0)$, the state as a function $\mathbf{x}(t, \mathbf{q})$ of finitely many control parameters $\mathbf{q} = (\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{N-1})$. Examples of possible parameterizations include but are not limited to piecewise constant, piecewise linear, continuous piecewise linear functions.

Once the control discretization is achieved and a numerical ODE solution is found, we obtain the following finite-dimension NLP:

$$\min_{\mathbf{q} \in \mathbb{R}^{nN}} \quad \int_0^T L(\mathbf{x}(t, \mathbf{q}), \mathbf{u}(t, \mathbf{q}))dt + \Phi(\mathbf{x}(T, \mathbf{q})) \tag{3.8}$$

subject to:

$$\begin{array}{rcll} \mathbf{g}(\mathbf{x}(t_i, \mathbf{q}), \mathbf{u}(t_i, \mathbf{q})) & = & \mathbf{0}, & i = 0, \ldots, N, \\ \mathbf{h}(\mathbf{x}(t_i, \mathbf{q}), \mathbf{u}(t_i, \mathbf{q})) & \geq & \mathbf{0}, & i = 0, \ldots, N, \\ \mathbf{r}(\mathbf{x}(0, \mathbf{q}), \mathbf{x}(T, \mathbf{q})) & = & \mathbf{0}. & \end{array} \tag{3.9}$$

Figure 3.2 shows an example of the control discretization and state numerical integration, using a piecewise constant control parameterization $\mathbf{q}$.

Single-shooting methods present several advantages: they only need an initial guess of the control parameterization $\mathbf{q}$, and they can rely on state-of-the-art ODE solvers to obtain the corresponding state $\mathbf{x}(t, \mathbf{q})$. Due to this fact, the underlying NLP problem has few degrees of freedom, even for large-scale ODE systems.

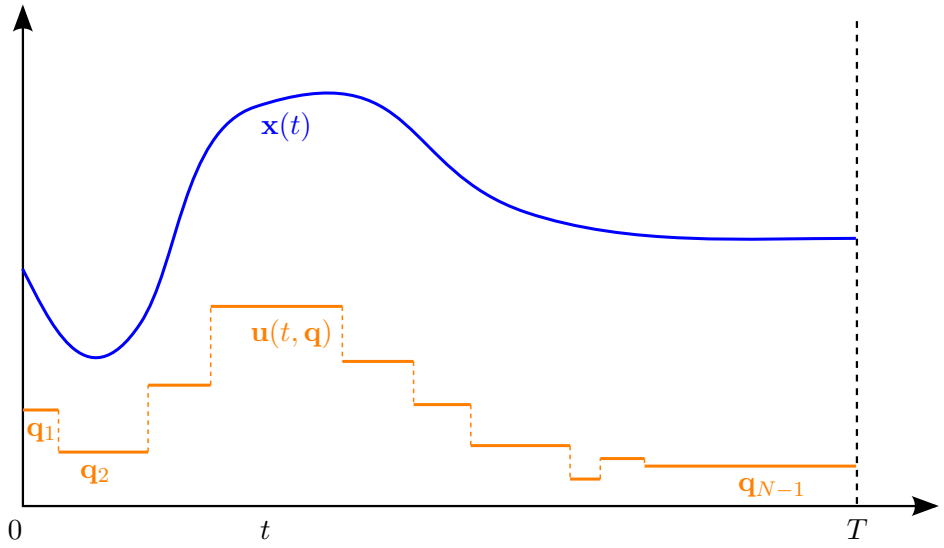


Figure 3.2: Solving the OCP with direct single-shooting methods: the controls are discretized on a coarse grid, and the state trajectory is found by numerical integration of the model ODE starting from an initial value. In this example, the discretized control $\mathbf{u}(t, \mathbf{q})$ is a piecewise constant function, and it is equal to $\mathbf{q}_i$ on sub-interval i.

However, because the state vector cannot be initialized, we cannot use the state knowledge to initialize it. This is problematic in tracking problems for instance, where we want to find an optimal policy starting from a good initial state trajectory. Furthermore, some ODE systems can be highly nonlinear and unstable (imagine a propelled rocket system); it can be very difficult to stabilize such systems over long trajectories and to make them achieve terminal constraints just by modifying $\mathbf{x}(0)$ and $\mathbf{q}$.

Direct Collocation

Direct collocation methods, as described in [Tsang 75], discretize both control and state vectors on a *fine* grid with node values $\mathbf{q}_i \approx \mathbf{u}(t_i)$ $\mathbf{s}_i \approx \mathbf{x}(t_i)$ respectively. This allows replacing the infinite ODE by finitely many equality constraints, and approximating the Lagrangian term in the objective function. Using forward differentiation for instance would give:

$$\begin{aligned} \dot{\mathbf{x}}(t) - \mathbf{f}(\mathbf{x}(t)), \mathbf{u}(t)) &= \mathbf{0}, \qquad t \in [0, T] \\ &\Downarrow \quad \text{forward differentiation} \\ \mathbf{c}_i(\mathbf{q}_i, \mathbf{s}_i, \mathbf{s}_{i+1}) = \frac{\mathbf{s}_{i+1} - \mathbf{s}_i}{t_{i+1} - t_i} - \mathbf{f}\left(\frac{\mathbf{s}_i + \mathbf{s}_{i+1}}{2}, \mathbf{q_i}\right) &= \mathbf{0}, \qquad i = 0, 1, \ldots, N-1 \end{aligned} \tag{3.10}$$

and

$$\begin{array}{c} \displaystyle\int_0^T L(\mathbf{x}(t),\mathbf{u}(t))dt \\ \Downarrow \\ \displaystyle\sum_{i=0}^{N-1} l_i(\mathbf{q}_i,\mathbf{s}_i,\mathbf{s}_{i+1}) = \sum_{i=0}^{N-1} L\left(\frac{\mathbf{s}_i+\mathbf{s}_{i+1}}{2},\mathbf{q_i}\right)(t_{i+1}-t_i) \end{array} \tag{3.11}$$

Using this discretization, we obtain a large but sparse NLP. It can be solved using efficient SQP or IPM solvers which are specialized for sparse problems, such as SNOPT [Gill 02] and IPOPT [Biegler 09]:

$$\min_{\mathbf{s}\in\mathbb{R}^{nN},\mathbf{q}\in\mathbb{R}^{nN}} \quad \sum_{i=0}^{N-1} l_i(\mathbf{q}_i,\mathbf{s}_i,\mathbf{s}_{i+1}) + \Phi(\mathbf{s}_N) \tag{3.12}$$

subject to:

$$\begin{array}{rcll} \mathbf{c}_i(\mathbf{q}_i,\mathbf{s}_i,\mathbf{s}_{i+1}) & = & \mathbf{0}, & i=0,\ldots,N, \\ \mathbf{g}(\mathbf{s}_i,\mathbf{q}_i) & = & \mathbf{0}, & i=0,\ldots,N, \\ \mathbf{h}(\mathbf{s}_i,\mathbf{q}_i) & \geq & \mathbf{0}, & i=0,\ldots,N, \\ \mathbf{r}(\mathbf{s}_0,\mathbf{s}_N) & = & \mathbf{0}. & \end{array} \tag{3.13}$$

State Parameterization One particular case occurs when the control and state derivatives can be directly derived from the states. There is thus no need to discretize the controls, which are here dependent variables, and only the states are discretized. The states can then be represented by smooth functions such as polynomials or splines [Sirisena 81]. This approach offers the advantage of leading to a NLP with a smaller number of variables than in the general case of direct collocation, and is quite common in robotics.

To conclude, direct collocation methods transcribe the OCP into a large-scale, but very sparse NLP, which can be solved by efficient solvers. It can treat unstable systems well, offering robust handling of path and terminal constraints. Furthermore, the state discretization is such that convenient state trajectories can be used as an initial guess, which is not the case for single-shooting methods. However, not all ODE solvers can be used as some of the most efficient ones use an adaptive-step scheme to perform state integration; this indeed requires changing the discretization grid during the optimization process, and leads to a change in the NLP dimensions.

Direct Multiple-Shooting

Direct multiple-shooting methods [Bock 84] were devised as hybrid methods between single-shooting and collocation methods; they are based on a coarse discretization of the control vector $\mathbf{u}(t) = \mathbf{q}_i$ for $t \in [t_i, t_{i+1}]$, and the addition of initial state values $\mathbf{s}_i$ for the state vector. These nodes serve as initial values for the numerical integration of the ODE over each sub-interval $[t_i, t_{i+1}]$:

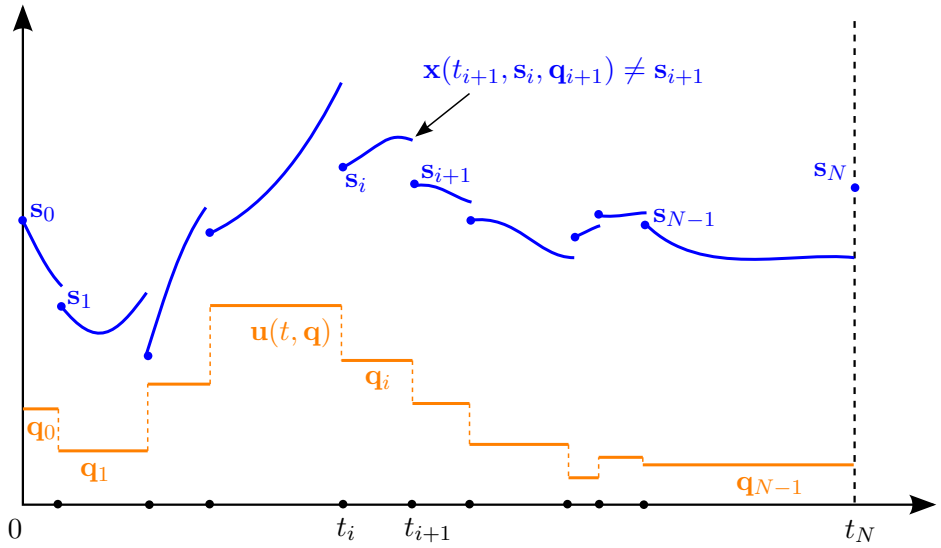


Figure 3.3: Solving the OCP with direct multiple-shooting methods: controls are discretized on a coarse grid, and several initial values for the state are given. The state trajectory is computed on each sub-interval by numerical integration of the model ODE starting from the node values. Note that the whole trajectory is not continuous, and additional continuity constraints need to be added.

$$\begin{array}{rcll} \dot{\mathbf{x}}_i(t, \mathbf{s}_i, \mathbf{q_i}) & = & \mathbf{f}(\mathbf{x}_i(t, \mathbf{s}_i, \mathbf{q}_i), \mathbf{q}_i), & t \in [t_i, t_{i+1}]\,, \\ \mathbf{x}_i(t, \mathbf{s}_i, \mathbf{q}_i) & = & \mathbf{s}_i. & \end{array} \tag{3.14}$$

Similarly, the Lagrangian term can be numerically integrated over each sub-interval:

$$l_i(\mathbf{s}_i, \mathbf{q}_i) = \int_{t_i}^{t_{i+1}} L(\mathbf{x}_i(t_i, \mathbf{s}_i, \mathbf{q}_i), \mathbf{q}_i) dt \tag{3.15}$$

Trajectory pieces $\mathbf{x}_i(t, \mathbf{s}_i, \mathbf{q}_i)$ are then generated, as shown in Figure 3.3. We can see that the whole trajectory is not necessarily continuous; we introduce continuity conditions to ensure state continuity over the whole duration, and the obtained finite-dimensional NLP is:

$$\min_{\mathbf{s} \in \mathbb{R}^{nN}, \mathbf{q} \in \mathbb{R}^{nN}} \quad \sum_{i=0}^{N-1} l_i(\mathbf{s}_i, \mathbf{q}_i) + \Phi(\mathbf{s}_N) \tag{3.16}$$

subject to:

$$\begin{array}{rcll} \mathbf{x}_i(t_{i+1}, \mathbf{s}_i, \mathbf{q}_i) - \mathbf{s}_{i+1} & = & \mathbf{0}, & i = 0, \ldots, N-1, \\ \mathbf{g}(\mathbf{s}_i, \mathbf{q}_i) & = & \mathbf{0}, & i = 0, \ldots, N, \\ \mathbf{h}(\mathbf{s}_i, \mathbf{q}_i) & \geq & \mathbf{0}, & i = 0, \ldots, N, \\ \mathbf{r}(\mathbf{s}_0, \mathbf{s}_N) & = & \mathbf{0}. & \end{array} \tag{3.17}$$

Let us summarize all variables $\mathbf{s}_i$ and $\mathbf{q}_i$ in one single vector as:

$$\mathbf{w} = (\mathbf{s}_0, \mathbf{q}_0, \mathbf{s}_1, \mathbf{q}_1, \ldots, \mathbf{s}_N) \in \mathbb{R}^{2nN}, \tag{3.18}$$

and let us gather continuity (including boundary), equality and inequality constraints in three vectors $\mathbf{C}$,$\mathbf{G}$ and $\mathbf{H}$ respectively. We can then rewrite the NLP problem as:

$$\min_{\mathbf{w}\in\mathbb{R}^{2nN}} F(\mathbf{w}) \quad \text{such that} \begin{cases} \mathbf{C}(\mathbf{w}) &= \mathbf{0}, \\ \mathbf{G}(\mathbf{w}) &= \mathbf{0}, \\ \mathbf{H}(\mathbf{w}) &\geq \mathbf{0}. \end{cases} \tag{3.19}$$

This NLP can be solved iteratively using an SQP method, where each step consists in building the approximate QP subproblem around the current iterate, as seen in Section B.3.1. The QP subproblem is written as:

$$\min_{\mathbf{p}\in\mathbb{R}^{2nN}} \frac{1}{2}\mathbf{p}^\top \nabla^2_{\mathbf{ww}}\mathcal{L}\mathbf{p} + \nabla F^\top \mathbf{p}, \quad \text{such that} \begin{cases} \mathbf{C} + \nabla\mathbf{C}^\top\mathbf{p} &= \mathbf{0}, \\ \mathbf{G} + \nabla\mathbf{G}^\top\mathbf{p} &= \mathbf{0}, \\ \mathbf{H} + \nabla\mathbf{H}^\top\mathbf{p} &\geq \mathbf{0}, \end{cases} \tag{3.20}$$

where $\mathbf{p} = (\Delta\mathbf{s}_0, \Delta\mathbf{q}_0, \Delta\mathbf{s}_1, \Delta\mathbf{q}_1, \ldots, \Delta\mathbf{s}_N)$, and $\mathcal{L}$ is the Lagrangian of F.

If we give a closer look at the first equality constraint, we notice that each block-component can be written as:

$$\begin{gathered} \mathbf{c}_i + \mathbf{X}^{\mathbf{s}}_i\Delta\mathbf{s}_i + \mathbf{X}^{\mathbf{q}}_i\Delta\mathbf{q}_i - \Delta\mathbf{s}_{i+1} = \mathbf{0}, \quad i = 0, \ldots, N-1 \\ \Updownarrow \\ \Delta\mathbf{s}_{i+1} = \mathbf{c}_i + \mathbf{X}^{\mathbf{s}}_i\Delta\mathbf{s}_i + \mathbf{X}^{\mathbf{q}}_i\Delta\mathbf{q}_i, \quad i = 0, \ldots, N-1, \end{gathered} \tag{3.21}$$

where $\mathbf{X}^{\mathbf{s}}_i$ and $\mathbf{X}^{\mathbf{q}}_i$ are the Jacobians of $\mathbf{x}(t_{i+1}, \mathbf{s}_i, \mathbf{q}_i)$ with respect to $\mathbf{s}$ and $\mathbf{q}$ respectively.

This leads to two conclusions: first, the continuity constraint vector Jacobian $\mathbf{C}^\top$ is block-sparse (it can be similarly shown that it is also the case for the other constraints gradients and the Lagrangian Hessian). Second, there exists a recurrence relation between $\Delta\mathbf{s}_{i+1}$ and $\Delta\mathbf{s}_i$ for $i = 0, \ldots, N-1$. This gives way for a *condensing strategy*, where the variables $\Delta\mathbf{s}_i, i = 1, \ldots, N$ are eliminated from the KKT system, and a reduced and simpler system is solved for the vector:

$$\mathbf{w}' = (\Delta\mathbf{s}_0, \Delta\mathbf{q}_0, \Delta\mathbf{q}_1, \ldots, \Delta\mathbf{q}_{N-1}) \tag{3.22}$$

The eliminated variables can then be reconstructed, starting from $\Delta\mathbf{s}_0$, using the recurrence relation from Equation (3.21).

The condensing strategy allows then to make advantage of the QP subproblem sparsity, and leads to an efficient SQP strategy. Such a method can be found in the MUSCOD-II [Leineweber 03a, Leineweber 03b] and ACADO [Houska 10] multiple-shooting optimal control solvers.

To conclude, direct multiple-shooting methods offer several advantages: they rely on a coarse discretization of both control and state vectors, using adaptive-step ODE solvers to integrate the state on the sub-intervals. They can thus use knowledge of the state at initialization, which makes them very suitable for applications where a good initial guess of the state can be given. Furthermore, the multiple-shooting scheme allows robust handling of all constraints, and can be potentially easy to parallelize. While its underlying NLP is not as sparse as in direct collocation methods, multiple-shooting methods are particularly powerful thanks to the condensing strategy, which takes advantage of the QP subproblem sparsity, and is used to solve them efficiently.

3.3.4 Non-Jacobian-Based Optimal Control

Note that for all methods we described above, the objective, dynamics and constraint functions can be nonlinear. They must be, however, at least C^1 so that the solvers can get an idea of the function local shapes and know where to look for the minimizer while obeying the constraints. This requirement can be alleviated by the use of non gradient-based optimal control methods.

We described a random optimization method in Section 1.1.3. It is a shortcut heuristic, and can be seen as an optimal control method which does not need the function Jacobians. If shortcut heuristics are applied in the state space, the resulting trajectory has a lower cost than the original one. However, no solution can be found outside the bounding box of the original trajectory due to the fact that the iterative process picks points to shortcut that are on the trajectory. Shortcut methods get easily stuck in local minimizers.

Another example is RRT* [Karaman 11]: it is a variant of the RRT sampling-based planner that relies on a simultaneous exploration of the configuration space and rewiring of the exploration tree so that it contains only low-cost connections. This exploration-rewiring iterative process continues even after one solution has been found, and it offers the interesting property of asymptotic convergence towards the *global* minimum-cost collision-free path. It is also shown that, in practice, the running time until an acceptable minimizer is reached is greater than the time needed to find any collision-free solution by only a constant factor. Of course, if we want to generate an optimal trajectory, we need to explore not only the configuration space $\mathcal{CS}$, but at least the whole state space $\mathcal{SS}$ (if not the space of both states and controls), where an element of $\mathcal{SS}$ is $\mathbf{x}$. In this case, the exploration might become too large to have tractable performance. Another problem arises from choosing the correct metric for choosing nearest neighbors, as well as the local optimal steering method. In [Perez 12], the authors propose a variant which uses the local linearization of a system to derive both coherent metric and extension method. Results have so far been only obtained on simple systems, and this method has yet to be tested on complex ones. Nevertheless, RRT* is an interesting approach as only one algorithm is needed to achieve global optimal motion planning.

3.4 Anthropomorphic System Dynamics

If we want to be able to generate feasible motions for anthropomorphic systems, we need to take into account their dynamics in the OCP formulation. Such underactuated systems are modeled by a rigid body kinematic tree attached to a floating base, as mentioned in Section 1.2.1; therefore a motion feasibility cannot be guaranteed unless dynamic balance constraints are enforced over the whole trajectory duration. We give in this section a brief overview about a floating-base system dynamics computation and the dynamic balance conditions. A complete description of state-of-the-art efficient dynamics algorithms can be found in [Featherstone 08].

3.4.1 Expressing Dynamics with Spatial Algebra

Usually, rigid body dynamics are written using 3D vector notations which keeps the translation and rotation parts of dynamic quantities apart: linear velocities vs angular velocities, forces vs torques, linear momentum vs angular momentum, etc. We introduce here 6D *spatial vectors* and their associated *spatial algebra* [Featherstone 08]: they allow to have a compact representation of dynamic quantities, which leads to increased performance of their associated operators.

For instance let us assume that we have a rigid body $\mathcal{B}$ with two coordinate systems A and B; each of them has an associated Cartesian frame and coordinates system. Let ${}^{A}\mathbf{v}$ and ${}^{A}\boldsymbol{\omega}$ denote the body linear and angular velocity in A coordinates. We have then the following relations:

$$\begin{array}{rcl} {}^{B}\mathbf{v} & = & \mathbf{E}({}^{A}\mathbf{v} - \mathbf{r} \times {}^{A}\boldsymbol{\omega}) \\ {}^{B}\boldsymbol{\omega} & = & \mathbf{E}\,{}^{A}\boldsymbol{\omega} \end{array} \tag{3.23}$$

where $r = \overrightarrow{AB}$ is the translation vector between the two coordinate systems, and $\mathbf{E} = {}^{B}\mathbf{R}_{A}$ is the rotation matrix from A to B.

Let us define the 6D spatial velocity vector:

$${}^{A}\mathbf{v}^{s} = \begin{pmatrix} {}^{A}\boldsymbol{\omega} \\ {}^{A}\mathbf{v} \end{pmatrix} \tag{3.24}$$

We can then rewrite Equation (3.23) as:

$${}^{B}\mathbf{v}^{s} = {}^{B}\mathbf{X}_{A}\,{}^{A}\mathbf{v}^{s}, \quad {}^{B}\mathbf{X}_{A} = \begin{pmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{E}\mathbf{r}\times & \mathbf{E} \end{pmatrix} \tag{3.25}$$

The same relation holds for both spatial velocities and accelerations, which are both referred to by the term *spatial motions*. Similarly, we can define a coordinate transformation relation for spatial forces:

$${}^{B}\mathbf{f}^{s} = {}^{B}\mathbf{X}^{*}_{A}\,{}^{A}\mathbf{f}^{s}, \quad {}^{A}\mathbf{f}^{s} = \begin{pmatrix} {}^{A}\mathbf{n} \\ {}^{A}\mathbf{f} \end{pmatrix}, \quad {}^{B}\mathbf{X}^{*}_{A} = {}^{B}\mathbf{X}^{\top}_{A} = \begin{pmatrix} \mathbf{E} & -\mathbf{E}\mathbf{r}\times \\ \mathbf{0} & \mathbf{E} \end{pmatrix} \tag{3.26}$$

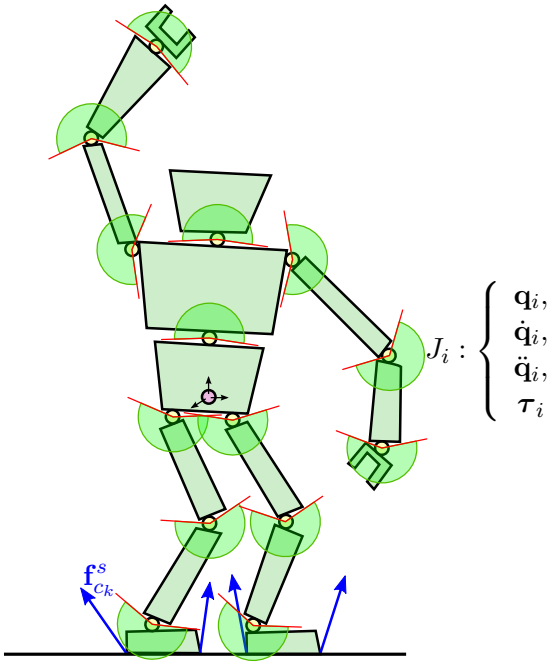


Figure 3.4: Non-vanishing and non-sliding contact forces are applied on the anthropomorphic system by its environment, and they are integrated in its dynamics equation. Joints limits are shown in red.

Note that except for the ordering of the translation and rotation components, spatial vectors represent the same concepts as velocity, acceleration and force wrenches, i.e. they both give a compact representation of a vector field. In what follows, we only use the spatial algebra notations.

In conclusion, the spatial vector notation provides a compact rewriting of the dynamics equations, which leads to efficient dynamics computation algorithms.

3.4.2 Dynamics Equation

By using a Newtonian dynamics formulation, we can express the compact dynamics equation for a floating-base robot with respect to the generalized coordinate and actuated torque vectors $\mathbf{q}$ and $\boldsymbol{\tau}$:

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \sum_k \mathbf{J}^\top_{c_k} \mathbf{f}^s_{c_k} = \mathbf{S}^\top \boldsymbol{\tau}, \tag{3.27}$$

where $\mathbf{A}$ is the *joint-space inertia matrix*, $\mathbf{b}$ is the joint-space bias force which account for both Coriolis and gravity effects, $\mathbf{J}_{c_k}$ is the *contact Jacobian* for the contact c_k that is submitted to spatial contact force $\mathbf{f}^s_{c_k}$. In this work, we assume that all bodies are rigid and that contacts are non-sliding. This leads to the necessary condition that each 3D contact force applied at a point must lie inside the positive *Coulomb friction cone* [Trinkle 97] defined by the inequalities:

$$\begin{aligned} \mathbf{f}_{c_k}^{\top}\mathbf{u}_{c_k} &\geq 0, \quad k = 1, \ldots, n_c, \\ \left\|\mathbf{f}_{c_k}^{\angle}\right\| &\leq \mu_s \left\|\mathbf{f}_{c_k}^{\perp}\right\|, \quad k = 1, \ldots, n_c, \end{aligned} \tag{3.28}$$

where $\mathbf{u}_{c_k}$ denotes the normal vector to the contact surface, $\mathbf{f}^{\perp}$ and $\mathbf{f}^{\angle}$ denote the normal and tangential contact forces to the contact surface respectively, and μ_s denotes the *limiting coefficient of static friction*, which depends of the contact interface nature (materials, temperature, etc.).

In addition to the dynamics equation, both of the actuators and the kinematic structure can impose several limitations, such as joint position limits

$$\underline{\mathbf{q}} \leq \mathbf{q} \leq \overline{\mathbf{q}}, \tag{3.29}$$

joint velocity limits

$$\underline{\dot{\mathbf{q}}} \leq \dot{\mathbf{q}} \leq \overline{\dot{\mathbf{q}}}, \tag{3.30}$$

and actuator torque limits

$$\underline{\boldsymbol{\tau}} \leq \boldsymbol{\tau} \leq \overline{\boldsymbol{\tau}} \tag{3.31}$$

These additional constraints, shown in Figure 3.4 will also have to be taken into account in the OCP formulation in order to generate feasible motions.

3.4.3 Inverse Dynamics

When the generalized position, velocity and acceleration vectors are known, we can use the dynamics equation 3.27 to retrieve the unknown actuated torques τ. This is a problem of *inverse dynamics*; the *Recursive Newton-Euler Algorithm (RNEA)* is an efficient algorithm which relies on a loop of forward propagation of spatial velocities and accelerations starting from the floating base, then a second loop of backward propagation of torques and forces starting from the leaves of the kinematic tree until the floating base. Its complexity is $\mathcal{O}(n)$ and it has a low count of operations, which makes it very suitable for optimal control applications.

3.4.4 Forward Dynamics

In *Forward dynamics*, on the opposite of the inverse dynamics problem, the generalized position, velocity and actuated torque vectors are known, and the accelerations $\ddot{\mathbf{q}}$ are unknown.

The *Composite Rigid Body Algorithm (CRBA)* is an efficient algorithm for computing the joint-space inertia matrix $\mathbf{A}(\mathbf{q})$. Interestingly, the bias force $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$ can be retrieved by computing the inverse dynamics while setting both accelerations $\ddot{\mathbf{q}}$ and contact forces $\mathbf{f}_{c_k}^{s}$ to $\mathbf{0}$. We can then use standard linear algebra solvers to solve the following system for $\ddot{\mathbf{q}}$:

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{S}^\top \boldsymbol{\tau} - \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) - \sum_k \mathbf{J}_{c_k}^\top \mathbf{f}_{c_k}^s \tag{3.32}$$

The whole algorithm complexity is $\mathcal{O}(n^3)$; the *Articulated Body Algorithm (ABA)*, just like RNEA, is based on a propagation method and offers a complexity of $\mathcal{O}(n)$. As the torques usually constitute the physical controls of a system, forward dynamics algorithms are very useful in simulation and control applications.

3.4.5 Dynamic Balance for Anthropomorphic Systems

Dynamic balance is a necessary condition to the generation of safe and feasible motions. We describe here possible ways of including this condition in an OCP.

Dynamic Balance with Zero-Moment Point

As we previously introduced the ZMP in Section 1.3.1, we simply recall that it provides a simple dynamic balance condition, namely that the ZMP must remain inside the contact support polygon, as long as the humanoid robot is moving on a flat floor. We saw in Section 1.3.2 how it can be derived for the cart-table model, and we briefly describe here a method to compute it for whole-body motions of an anthropomorphic system using inverse dynamics.

Let $\mathbf{q}$,$\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ denote respectively the rigid-body system generalized position, velocity and acceleration vectors. We recall that, if feasible contact forces are applied to its bodies, a necessary and sufficient condition for a humanoid robot dynamical balance is that it can realize the state $\mathbf{q}$,$\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ using only its physical actuators. As the humanoid robot does not have any thrusters, this condition is equivalent to requiring that the 6D generalized torque applied by the floating joint be 0:

$$\boldsymbol{\tau}_{\boldsymbol{fl}} = \mathbf{0}, \tag{3.33}$$

where $\boldsymbol{\tau}_{\boldsymbol{fl}}$ denotes the floating-joint torque vector, see [Hirukawa 06].

Conversely, if we set all contact forces to zero and we compute the joint generalized torques using the RNEA, $\boldsymbol{\tau}_{\boldsymbol{fl}}$ will be equivalent to a spatial force (or wrench): it is the force that the floating joint would need to exert in order to make the whole system realize the state given by $\mathbf{q}$,$\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$. This spatial force, expressed in the floating base frame and denoted ${}^{fl}\mathbf{f}_c^s$, is thus equal to the resultant spatial force that includes all the necessary contact forces. Computing the ZMP is then a simple matter of transforming ${}^{fl}\mathbf{f}_c^s$ to the absolute world frame W, and finding the coordinates (p_x, p_y) of the point on the floor such that the moments around the x and y axes, i.e. the first two components of ${}^{W}\mathbf{f}_c^s$, are equal to zero. This point $\mathbf{p}$ is, according to its definition, the Zero-Moment Point which we are looking for. We can then guarantee a humanoid robot dynamic balance by ensuring $\mathbf{p}$ stays inside the support polygon $\mathcal{P}_{sup}$, i.e. the convex hull of all contact points.

Dynamic Balance with Non-Coplanar Contact Forces

Let us briefly mention that the ZMP criterion is extended to handle non-coplanar contact points such as arms and hands in [Harada 03], and is called the *Generalized ZMP (GZMP)*. An associated support polyhedron is defined, and the GZMP must stay inside this polyhedron to ensure the dynamic balance of the robot. Although this approach is interesting, it is not as simple as the ZMP criterion, and we prefer to deal directly with contact forces through the complete dynamics of the robot.

Forward Dynamics We present here a method that ensures the dynamic balance constraint for optimal control, as seen in [Mombaur 05]. It relies on forward dynamics, where the system is torque-controlled. The state contains both the acceleration $\ddot{\mathbf{q}}$ and contact forces $\mathbf{f}^s_{c_k}$.

As we work under the assumption that a contact point does not move during the motion, we have:

$$\mathbf{p}_{c_k}(t) = \mathbf{p}_{c_k}(0), \quad k = 1, \dots, n_c, \tag{3.34}$$

where $\mathbf{p}_{c_k}$ denotes the 6D position of contact c_k.

By derivating it twice we obtain the contact condition:

$$\mathbf{J}_{c_k}\ddot{\mathbf{q}} + \dot{\mathbf{J}}_{c_k}\dot{\mathbf{q}} = \mathbf{0}, \quad k = 1, \dots, n_c, \tag{3.35}$$

Let $\mathbf{f}^s_c$ and $\mathbf{J}_c$ denote the concatenated contact force vector and contact Jacobian respectively:

$$\mathbf{f}^s_c = \begin{pmatrix} \mathbf{f}^s_{c_1} \\ \vdots \\ \mathbf{f}^s_{c_k} \\ \vdots \\ \mathbf{f}^s_{c_{n_c}} \end{pmatrix}, \qquad \mathbf{J}_c = \begin{pmatrix} \mathbf{J}_{c_1} \\ \vdots \\ \mathbf{J}_{c_k} \\ \vdots \\ \mathbf{J}_{c_{n_c}} \end{pmatrix} \tag{3.36}$$

The following linear system can then be built using the dynamics equation and contact condition, and solved for $\ddot{\mathbf{q}}$ and all $\mathbf{f}^s_{c_k}$.

$$\begin{pmatrix} \mathbf{A} & \mathbf{J}_c^\top \\ \mathbf{J}_c & \mathbf{0} \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}} \\ \mathbf{f}^s_c \end{pmatrix} = \begin{pmatrix} \mathbf{S}^\top \boldsymbol{\tau} - \mathbf{b} \\ -\dot{\mathbf{J}}_c \dot{\mathbf{q}} \end{pmatrix} \tag{3.37}$$

Note that $\mathbf{J}_c$ must be full-rank, otherwise, the system is not invertible. Therefore, this approach allows finding a unique set of spatial forces as long as no more than one spatial contact force is exerted on each rigid body. If we consider the humanoid walking case, this means that one spatial contact force is applied on each foot. However, the sum of the two solution forces can lead to a ZMP lying outside the support polygon, as the foot geometry is never taken into account. Solving the previous linear system is unfortunately not always sufficient to ensure dynamic balance; additional constraints, such as ZMP constraints [Koch 12], need to be added.

Inverse Dynamics We saw in Equation (3.33) that if we apply feasible contact forces and the floating-joint torques are zero, then the humanoid robot is dynamically balanced. If we chose to control our system with the generalized acceleration vector $\ddot{\mathbf{q}}$ and contact forces $\mathbf{f}_{c_k}^s$, we can use a simple integration scheme to retrieve the generalized position and velocity vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$, compute the torques $\boldsymbol{\tau}_{fl}$, and add Equation (3.33) as an equality constraint in the OCP formulation. This gives a way of integrating the dynamics and enforcing the dynamic balance of the humanoid robot through a 6D vector equality constraint, given any set of contact forces, even when some of them are exerted on the same bodies. In order to ensure that only feasible contact forces are applied, the inequalities from Equation (3.28) also need to be taken into account. A similar formulation, where the contact forces are considered as optimization variables, was suggested in [Saab 12].

3.5 (Self-)Collision Avoidance

Beside generating dynamically feasible motions for an anthropomorphic system, we need to make sure no collision occurs during the motion. Let us assume a collision-free initial guess is fed to the OCP solver. The optimization solver iteratively reshapes the motion, and either collisions between the robot and itself or collisions with the environment might ensue. This motivates the need to take into account (self-)collision avoidance in the OCP.

3.5.1 Distance Pairs

Consider a body B_i of the robot. Figure 3.5 gives an idea of the potential complexity of self-collision avoidance: if the robot has N_B bodies, there are $N_B - 1$ potential pairs of bodies $\langle B_i, B_j \rangle$ that need to be checked at a given configuration $\mathbf{q}$ in order to ensure that there is no collision between the robot bodies. Since the pair $\langle B_i, B_j \rangle$ is equivalent to the pair $\langle B_j, B_i \rangle$, the total number of self-collision body pairs can be as high as to $\frac{N_B(N_B-1)}{2}$. Furthermore, assuming that the obstacles surrounding the robot are considered as a single geometric entity O, we also need to check N_B body-environment pairs $\langle B_i, O \rangle$ for collisions.

One could use efficient collision detection algorithms, but their return result is Boolean and is not suitable for gradient-based optimal control. In order to guarantee (self-)collision avoidance, we need to compute the distance between for all potential collision pairs (or distance pairs), and make sure this distance is positive during the whole duration of the optimized motion.

3.5.2 Distance Computation for Collision Avoidance

A geometry can be represented accurately by a cloud of vertices, and a set of facets, usually triangles, which define the surface of the geometry. One could then express inequality constraints using the exact distance between the polyhedral geometries.

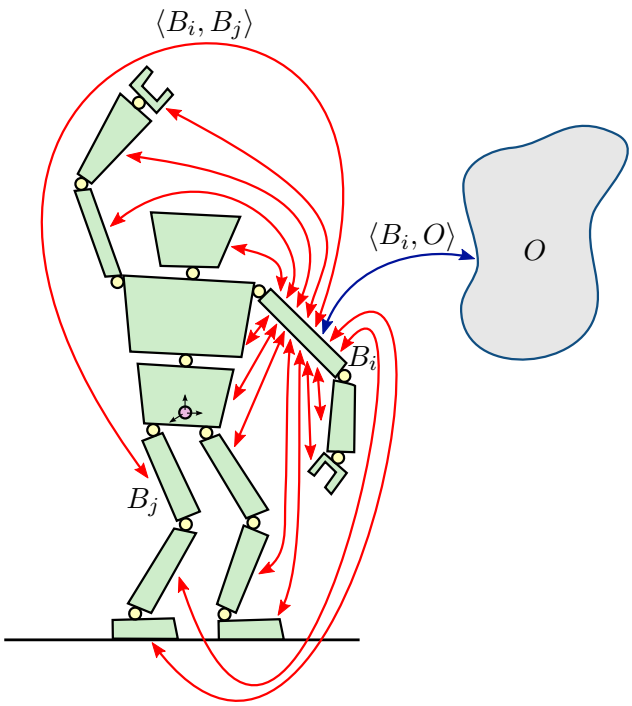


Figure 3.5: All self-collision possible pairs are shown in red for body B_i. The collision pair between B_i and an obstacle O is also shown in blue.

Thanks to bounding volume hierarchy representations of the polyhedra, the distance computation can be precise and relatively efficient [Larsen 00], especially when the algorithms are parallelized on graphics processing unit (GPU) [Lauterbach 10]. However, such algorithms return a constant zero distance when collisions occur, thus leading to a zero gradient; this can be very prohibitive as numerical optimization methods rely on the local constraint information to generate feasible iterates. In [Kim 02], a penetration computation algorithm is proposed, but computation times are still too restrictive.

One strategy to cope with poor performance and gradient discontinuities is to consider bounding volumes that contain the exact robot body geometries. As this is a conservative approach, some motions such as ones involving fine object manipulation cannot be generated. In most cases, however, this approach is suitable for a variety of robotic applications a constitutes a good trade-off between precision and computation time. In [Escande 07] a nice solution to this problem is described: they introduce *Sphere-Torus Patches Bounding Volumes (STPBV)*, which are strictly convex bounding volumes of the exact polyhedral body geometries; the returned distance can be negative for slight collisions, and it is shown that the distance between a STPBV and a convex mesh is always C^1. These constraints are successfully used for self-collision avoidance in posture optimization, and real-time control of a humanoid robot [Stasse 08]. Capsules, which are basically cylinders capped by half-

spheres, are slightly more conservative than STPBV, but offer an even simpler way of computing distances: computing the distance between two capsules is equivalent to computing the distance between two segments and subtracting the segment radii. In [Kanoun 11] for instance, such bounding volumes are successfully used to avoid self-collision for a humanoid robot in a real-time control application.

We discuss here an important aspect of OCP solutions when using direct methods, seen in Section 3.3.3. Indeed, these methods are based on a transcription of the OCP into a finite-dimensional NLP, through a grid discretization. Constraints are also discretized over the trajectory duration, which means that they are verified on the grid points, but not between them. In [Lee 12], based on previous work for safe trajectory optimization [Lengagne 13], capsules are used as bounding geometries for each body; the minimum distance over each sub-interval of the discretization grid is computed for all distance pairs, and this minimum distance is required to stay positive. Interestingly, this leads to a final optimized motion which is guaranteed to be collision-free for any $t \in [0, T]$. So far this method has been applied to implement self-collision avoidance, i.e. motions are generated in the absence of obstacles.

We briefly present the signed distance transform [Felzenszwalb 04], which is an alternative method for computing distances: assuming that the environment is static, an offline voxelized grid can be built, and the minimum distance to the obstacles is computed for each voxel. If there is a collision, the distance is negative and is equal to the penetration depth. This method allows fast distance computation for optimization algoritm, but the voxel grid needs to be fine enough to allow acceptable gradient computation through finite differentiation.

3.6 Optimal Control Applications for Anthropomorphic Systems

Over the past thirty years, optimal control techniques have been successfully applied in the fields of Robotics, Biomechanics and Computer Graphics. The problem of tracking a reference path while minimizing time can for instance be modeled as an OCP and solved thanks to state-of-the-art solvers, see [Bobrow 85, Verscheure 09, Suleiman 10]. In this case, the initial path is assumed to be collision-free; as solving the path tracking problem consists in applying a time parameterization without deforming the path, collision avoidance constraints are not taken into account.

Optimal control techniques can also be used for motion imitation such as in [Suleiman 08], where trajectories are generated for a humanoid robot by minimizing the error with reference human motions and ensuring its dynamic balance. Self-collisions are then post-processed when replaying the motion, using a task-based controller for instance, see [Kanehiro 08].

In *Model Predictive Control (MPC)*, the current control of a system is derived from its future states to ensure its stability. Finding the best control can be done through an optimal control formulation over a receding time horizon, in order to ensure maximum dynamic balance for instance. This has led to online control ap-

plications for a biped walking motion of a humanoid robot [Kajita 03, Herdt 10], and more complex locomotion on non-flat terrains for animated avatars [Coros 10, Tassa 12].

From a biomechanics point of view, some aspects of human motion can be represented as the resulting optimal policies of OCP. Conversely, solving the same OCP can allow human behaviors to emerge from the solutions: such behaviors include walking [Chevallereau 01], running [Schultz 10], emotional walking [Felis 12] and even more general motions involving seamless extreme locomotion and manipulation [Mordatch 12]. It is interesting here to point out the gradual convergence of the Robotics, Biomechanics and Computer Graphics fields as simulation models become more and more realistic and are able to take into account various kinds of physical interactions including gravity, actuator models, contact models, etc.

Some humanoid motions are very hard, or even impossible to generate without optimal control methods. Indeed, dynamic motions greatly expand the capabilities of humanoid robots which otherwise would have to satisfy the more restrictive quasi-static balance constraints. Complex motions such as ball-kicking [Miossec 06] and weight-lifting [Arisumi 08], and parkour [Dellin 12] can then be obtained. Also, the accessible space of a humanoid robot can be further increased if it uses its environment to help it achieve its goal; this involves generating multiple non-coplanar contact motion, and optimal control methods have proved to be very powerful tools to do so [Lengagne 13].

In many cases, robots have to move in the presence of obstacles. Therefore we must make sure that the resulting optimal policy does not lead to any collision between the robot and its environment or even itself. In [Dubowsky 86], time-optimal collision-free trajectories are computed for a 6-DoF manipulator by adding, to the OCP objective function, a penalty term that computes the distance to obstacles and allows avoiding them. Penalty terms are also used in CHOMP [Ratliff 09], an efficient optimal control solver which relies on a covariant gradient descent technique, and in STOMP [Kalakrishnan 11], a similar solver which relies on trajectory stochastic perturbations in order to find collision-free optimal trajectories without computing function Jacobians.

All previous applications involve simple robotic arms; in [Toussaint 07], a humanoid robot trajectory is modeled as a finite sequence of attractors in the task-space, such as a 3D hand position. This allows a great dimension reduction of the underlying NLP and leads to very short optimizations times of the order of a few seconds. An obstacle avoidance penalty term is also added in the objective function, and enables the robot to avoid both self-collisions and collision with simple planar obstacles. In this latter work, the initial trajectory can potentially generate collisions, although there is no guarantee that the optimization process will succeed in finding a collision-free solution in more complex environments. Furthermore, as no contact body position or dynamic balance tasks are considered, fast motions cannot be generated with this framework without compromising the robot dynamic balance.

In the general case, minimizing an objective function which contains constraints

transcribed as penalty terms allows building unconstrained optimal control problems. Good convergence rates can then be obtained as long as the problem dimension is small, and as long as the sum of too many penalty functions does not lead to an increased nonlinearity of the objective function. Otherwise, the performance gain obtained by solving an unconstrained NLP is neutralized by a poor convergence rate. Furthermore, the sum of penalty terms is usually weighted to give more or less importance to each term. This leads to the problem of choosing the correct weights such that the resulting trajectory is satisfactory for a variety of problems.

All methods cited above make profit from optimal control to achieve several applications of motion generation. However, current formulations either work under the assumption that the initial guess is collision-free, or allow it to be slightly in collision by means of linear interpolation between the initial and final states, without giving any guarantee that the optimization process will succeed. In fact, there are cases where the solver might get stuck in local minima and fail to generate a trajectory that avoids collisions of the robot with either itself or the environment.

The idea of combining planning algorithms with optimization methods is not new, and is especially motivated by the wide use of sampling-based planners which generally produce collision-free paths of poor-quality [LaValle 06]. The shortcut method, which we used in Chapters 1 and 2, can be seen as a local optimization method. In [Geraerts 07], several shortcut heuristics are developed to increase a path quality and clearance from obstacles, but they can be used to shorten a path length with respect to a given metric only in the configuration space as time is not considered. A suitable shortcut heuristic for optimal control of simple robotic arms is described in [Hauser 10b]: starting from an initial collision-free path given by a RRT path, an initial trajectory is placed on the path, making stops at the milestones to ensure it is still collision-free, then a shortcut heuristic is called several times to reduce the trajectory duration under bounded velocity and acceleration constraints, while making sure the result is still collision-free. Similarly, a method that combines a sampling-based planner and the STOMP optimizer is proposed, see [Mainprice 12]. Note that in both previous methods, results were successfully obtained for simple 7-DoF arms; as they do not allow enforcing complex constraints such as closed-loop constraints or dynamic balance, achieving optimal motion planning for anthropomorphic systems with such methods does not seem to be possible.

3.7 Contribution

To our best knowledge there is no available algorithmic approach that addresses the global problem of optimal motion planning for complex robots, such as anthropomorphic systems, in the presence of arbitrarily complex obstacles.

We therefore propose a new framework for optimal motion planning. Given a humanoid robot geometric and dynamic model, an exact model of the environment,

start and end configurations, and a robot contact stance, we first *plan* a collision-free *statically balanced* path that satisfies all kinematic constraints. We use the constrained RRT planner, which we presented in Section 1.1.2, to this end. We convert the path to an initial trajectory using a suitable time parameterization, and we then *optimize* it to generate a locally-optimal collision-free *dynamically-balanced* trajectory. The MUSCOD-II solver uses a direct multiple shooting method to solve the formulated optimal control problem. Note that this involves both finding a new time parameterization for the trajectory, and reshaping the path in a geometrical sense, so it is not simply a problem of optimal path tracking.

In order to ensure (self-)collision avoidance, we choose to model distance constraints using bounding capsules around the robot exact body geometries. In previous applications, the capsule parameters were set by hand by the user for a given body. We therefore provide an automatic bounding capsule generation tool; it relies on a NLP formulation that allows us to find the minimum-volume capsules around bodies which are modeled by polyhedra. The capsules allow us then to enforce collision-avoidance constraints between the robot, obstacles and itself.

To ensure dynamic balance during the robot motion, most of the results we present on the ZMP criterion; we introduce later on some preliminary results when using a multiple contact force formulation.

The full framework is successfully applied to generate optimal collision-free trajectories for a humanoid robot both in simulation and on the HRP-2.

The collision avoidance constraints are tackled in Section 3.8 and used in the optimal control problem described in Section 3.9. Section 3.10 showcases results obtained for the robot HRP-2. We also show an extension to multiple contact points in Section 3.11.

3.8 (Self-)Collision Avoidance Constraints

Before providing a description of our optimal motion planning framework, we first focus in this section on building the collision avoidance constraints through the automatic generation of bounding capsules around the robot body geometries, and the automatic pruning of unnecessary distance pairs $\langle B_i, B_j \rangle$ and $\langle B_i, O \rangle$.

3.8.1 Computing minimum bounding capsules

Capsule can be represented as the union of a cylinder and two half-spheres, or sphere-swept segment. It is uniquely determined by its two segment endpoints $\mathbf{e_1}$, $\mathbf{e_2}$ and its radius r. Capsules are convex geometries, and have been widely used in Robotics and Computer Graphics as they provide a simple representation of more complex polyhedral geometries.

In [Kanoun 11], the bounding capsules parameters are set by hand to obtain the best fitting capsules around the body geometries, which is not very practical especially if we want to use various robot geometric models. In [Eberly 07], a method

is proposed to compute a bounding capsule of a set of vertices, but not necessarily the best fitting one. This method is based on first determining the capsule segment direction using a least-squares regression, setting the capsule radius by finding the farthest vertex from the line, and finally determining the segment length by trying to make the capsule spherical caps come as close as possible to each other.

We propose here to automatically find minimum-volume capsule parameters by solving, offline and once for each body of the robot, the following optimization problem:

$$\min_{\mathbf{e_1},\mathbf{e_2},r} \quad \|\mathbf{e_2}-\mathbf{e_1}\|\pi r^2 + \frac{4}{3}\pi r^3 \tag{3.38}$$

subject to:

$$r - d(\mathbf{v},\mathbf{e_1}\mathbf{e_2}) \quad \geq \quad 0, \text{ for all } \mathbf{v} \in \mathcal{P}, \tag{3.39}$$

where $d(\mathbf{p},\mathbf{e_1}\mathbf{e_2})$ is the distance of $\mathbf{p}$ to line segment $\mathbf{e_1}\mathbf{e_2}$. Equations 3.38 and 3.39 mean we want to find the minimum-volume capsule while ensuring all vertices $\mathbf{v}$ of the underlying polyhedron $\mathcal{P}$ lie inside the capsule.

The problem we defined above is a NLP with inequality constraints. It can be solved using either SQP or IPM methods, as long as we give suitable initial parameters $\mathbf{e_1}_0$, $\mathbf{e_2}_0$, and r_0. Luckily, a good initial guess can be provided by the bounding capsule method we presented earlier, as all constraints are satisfied and the initial volume is not far off the optimal volume.

It can be proved that a convex geometry is a bounding volume of a set of points $\mathcal{P}$ if and only if it is a bounding volume of its convex hull $\mathcal{H}_\mathcal{P}$, which is characterized by a lower vertex count than $\mathcal{P}$. We can use this property to our advantage: by first computing the convex hull and finding the minimum-volume capsule for $\mathcal{H}_\mathcal{P}$, we are sure to have better optimization performance as the number of constraints is greatly reduced, especially for non-convex geometries.

We solve our minimization problem with RobOptim [Moulard 09, Moulard 12a] and the IPOPT solver [Biegler 09], and we use our implementation[1] to find the optimal bounding capsules for both HRP-2 and Romeo [Guizzo 10] humanoid robots. Table 3.1 demonstrates how solving optimization problem instances for $\mathcal{H}_\mathcal{P}$ instead of $\mathcal{P}$ accelerates their convergence. Figures 3.6 and 3.7 show the best fitting bounding capsules superimposed over the original robot geometries.

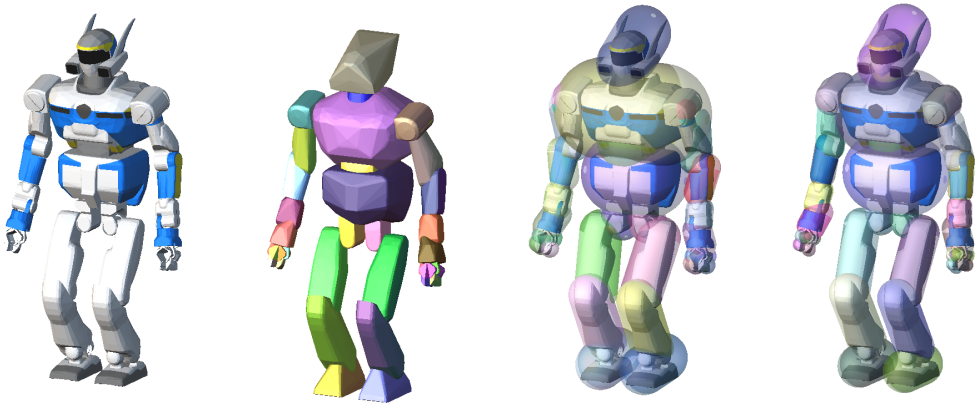
3.8.2 Computing Distances for Pairs

The optimal capsule parameters can now be used to compute distances for body-body and body-environment pairs. With respect to body-body distance pairs, we can compute their minimum distance by first computing the distance between the two capsule segments, then subtract their radii in order to obtain the real distance.

[1] An open-source implementation can be found at `https://github.com/roboptim/roboptim-capsule/` and `http://roboptim.net/roboptim-capsule/doxygen/1.0.1/`

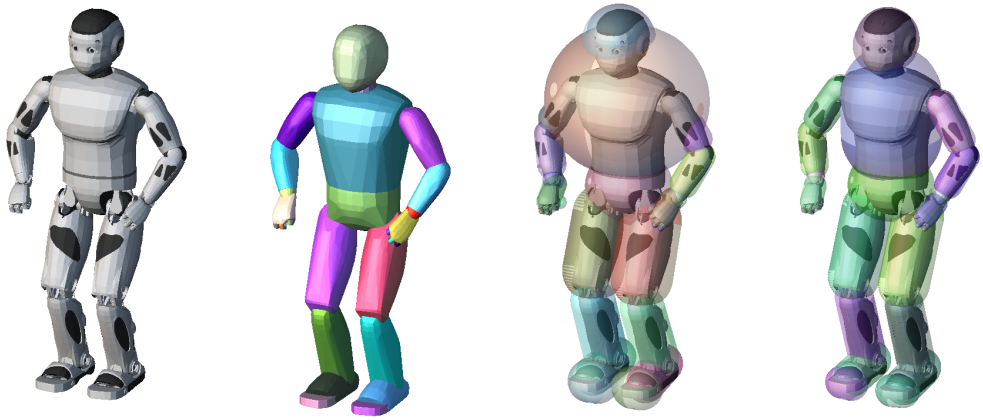
Table 3.1: Performance of minimum-volume bounding capsules generation.

Robot	Body count	Mean vertex count per body	Total computation time without convex hull (s)	Total computation time using convex hull (s)
HRP-2	41	1526	46.8	7.50
Romeo	46	7033	410	27.6



(a) Original geometries. (b) Convex hull. (c) Initial capsule parameter guess. (d) Optimization result.

Figure 3.6: Minimum-volume bounding capsule generation for the HRP-2.



(a) Original geometries. (b) Convex hull. (c) Initial capsule parameter guess. (d) Optimization result.

Figure 3.7: Minimum-volume bounding capsule generation for the Romeo robot.

We rely on the Wild Magic geometric library [Schneider 03, Eberly 11] to compute this distance in an average time of 2 μs.

Concerning capsule-environment pairs, as the environment is modeled by polyhedral meshes, we can compute their distance by computing the distance between the capsule segment and mesh, then subtract the capsule radius. State-of-the-art distance computation algorithms rely on building a hierarchical tree of simple bounding volumes around the mesh. In our work, we rely on an implementation of OBB-Trees in the Kineo Collision Detection (KCD) library to compute distances for capsule-environment pairs. For the environment in figure 3.8, the distance for one capsule-environment pair takes about 500 μs to be computed, since the environment is assumed to be perfectly modeled by polyhedron meshes. Note that even for very efficient implementations, the tree traversal scheme in bounding volume hierarchies implies running multiple proximity queries and we cannot hope for good performance unless GPU-based implementations are used.

3.8.3 Body Distance Pair Selection

We mentioned in Section 3.5 that if we were to take into account all body distance pairs of a robot, we could end up with $\frac{N_B(N_B-1)}{2}$ possible pairs. This means that for a robot like HRP-2 with 41 bodies, we can have up to 820 pairs and it can be very costly to evaluate the distance for all of them. Luckily, some bodies are either always colliding because they are adjacent in the kinematic tree, or never colliding due to kinematic constraints; for the particular example of the HRP-2, its kinematic tree and joint limits are such that the head body can never collide with its chest or any of its feet. The pairs corresponding to those bodies can therefore be safely pruned.

In order to avoid hand-checking of all pairs, we use the offline tool implemented in [Sucan 11]: it relies on finely exploring the configuration space, using a sampling-based planner such as RRT, and keeping track of colliding bodies. In the case of the HRP-2, this allows us to keep only 510 "useful" pairs out of 820 pairs. Although not used in this work, note that there exist online collision pair pruning techniques which allow to accelerate collision detections and proximity queries [Ericson 04].

As in all our examples, we consider the particular case of double-support motion, we can be sure that most of the leg bodies cannot collide with each other due to the additional kinematic constraints. This is a manual step, but it could be automated if additional kinematic constraints were taken into account in the previously described tool. We finally end up with 327 capsule-capsule pairs that must be all evaluated to guarantee self-collision avoidance.

Similarly, we can do more effort and prune some of the capsule-environment pairs that do not need to be checked due to the particular kinematic structure of a robot. In the case of HRP-2 for instance, if both waist and chest are not in collision with the environment, we can be sure that it will be the same for the intermediate body linking them. This case is not handled in the automated tool. Out of 41 potential

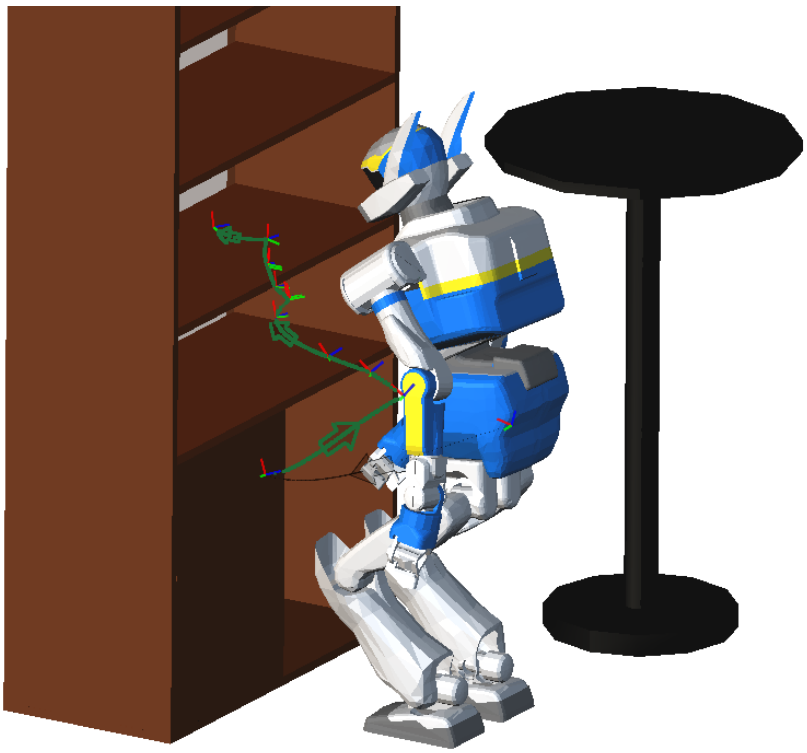


Figure 3.8: Path found by the path planner in a shelves environment.

capsule-environment pairs, we keep 23 pairs.

3.9 Optimal Motion Planning Framework

Now that we have defined collision avoidance constraints, we can build the full optimal motion planning framework. It can be decomposed into two main stages, namely constrained path planning and optimal control, with an intermediate stage which builds a time parameterization over the path that is generated by the planner.

In our work, we focus on generating collision-free trajectories for the HRP-2 in the case both its feet remain in the same position on a horizontal floor. We choose to minimize the integral, over a fixed duration, of the generalized square jerk $\dddot{\mathbf{q}}(t)$; this helps obtain smooth motions.

3.9.1 Constrained Path Planning

We use the constrained planner in [Dalibard 09], which we described in Section 2.1.2, and is implemented with the motion planning library KineoWorks^{TM}[Laumond 06]. This planner allows generating a collision-free path, while guaranteeing that the solution path lies on a manifold of the configuration space. We want to generate

for HRP-2 a collision-free path that guarantees its quasi-static balance when it is standing on both feet. We then define the manifold $\mathcal{M}$ with the following stack of equality constraints:

1. Right foot has a fixed 6D transformation,
2. Left foot has a fixed 6D transformation,
3. Center of mass vertical projection lies in the center of the support polygon.

Additionally, we would like to avoid choosing a single goal configuration $\mathbf{q}_g$, but instead define a goal task $\mathbf{x}_g$. This task can be defined by a sub-manifold $\mathcal{M}_g$ of the planning manifold $\mathcal{M}$. For a simple object manipulation task, $\mathcal{M}_g$ can be defined as the intersection between $\mathcal{M}$ and the manifold defined by the following stack of equality constraints:

1. Gripper has the same 3D position as the object to grab.
2. Gripper thumb is oriented vertically.

Given a start configuration $\mathbf{q}_s$ a planning manifold $\mathcal{M}$ and a goal sub-manifold $\mathcal{M}_g$, we first create a set of goal configurations $\mathbf{q}_g$ by sampling a fixed number of configurations in $\mathcal{M}_g$, then we solve the path problem from $\mathbf{q}_s$ to $\mathbf{q}_g$. The constrained planner diffuses trees from $\mathbf{q}_s$ and each configuration of $\mathbf{q}_g$, and stops once at least one of the goal configurations is in the same connected component as $\mathbf{q}_s$. A shortcut optimizer can then prune unnecessary waypoints and shorten the solution path. Figure 3.8 shows an example where HRP-2 has to grab an object on the lower shelf and place it on the upper shelf.

3.9.2 Time Parameterization for Initial Trajectory

The constrained path planner generates a collision-free statically balanced path where the kinematic constraints are enforced, but we still need to apply a time parameterization before feeding it to the optimal control solver. We want to minimize the sum of square jerks; it is shown in [Flash 85] that an unconstrained minimum-jerk trajectory is a polynomial of degree 5 which can be explicitly computed if the initial and final states are known. We choose then to place minimum-jerk trajectories between each pair of path waypoints, assuming they start and end at zero velocity and acceleration. This ensures that the configuration $\mathbf{q}(t)$ follows exactly the solution path and that collision avoidance constraints are not violated around the waypoints.

As we rely on a direct multiple-shooting optimal control solver, the minimum-jerk trajectories that have been computed are discretized along the state and control grid. Figure 3.9 shows an example where the time grid has 20 sub-intervals and a duration of 2 seconds, the control is a piecewise linear function representing the jerk,

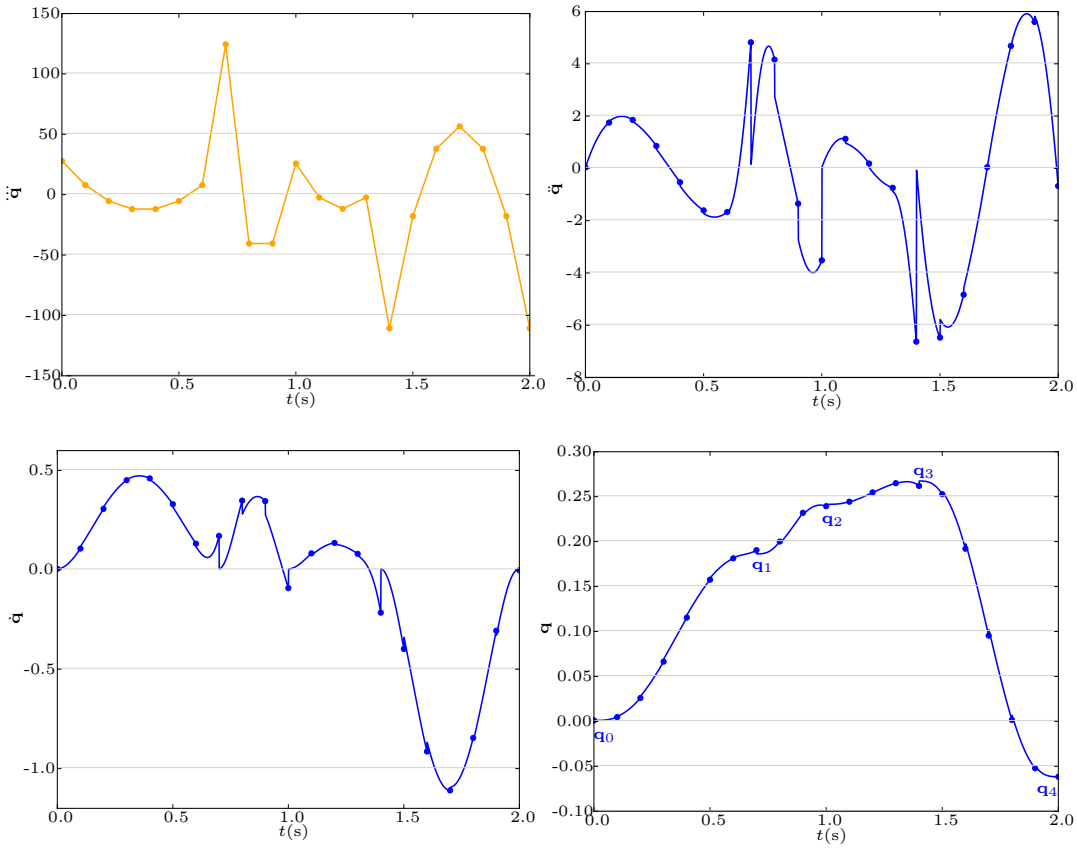


Figure 3.9: In this example, the planned path contains 5 milestones. The jerk (in orange) is a continuous piecewise linear function, and is computed such that the position trajectory starts and ends with zero velocity and acceleration at each milestone. The state nodes (blue dots) are similarly chosen, and the controls are integrated over each sub-interval to obtain the full trajectory (in blue). Note that discontinuities are observed because of the control parameterization.

and the state is comprised of the position, velocity and acceleration. Note that as only the nodes $\mathbf{s}_i$ are known, the remainder of the state trajectory can be obtained using the control representation and successive time integrations.

3.9.3 Optimal Control Problem Formulation

We can now build the full OCP formulation, and let the optimization solver reshape the initial guess in order to minimize the objective function while enforcing all constraints. This should lead to smooth motions without intermediate stops.

Objective function

We choose to minimize, for a fixed duration, the integral over time of the sum of square jerks, as this criterion leads to smooth trajectories.

The objective function can then be written as:

$$J = \int_0^T \dddot{\mathbf{q}}(t)^T \dddot{\mathbf{q}}(t) dt \tag{3.40}$$

and we define the state and control variables to be:

$$\begin{array}{rcl} \mathbf{x}(t) & = & [\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)]^T \\ \mathbf{u}(t) & = & [\dddot{\mathbf{q}}(t)]^T \end{array} \tag{3.41}$$

Equality and inequality constraints

Joint constraints Each actuated joint is subject to physical limitations of its underlying actuator and mechanical structure. Box constraints on angular, speed and torque limits are then added as:

$$\begin{array}{ccccc} \underline{\mathbf{q}} & \leq & \mathbf{q} & \leq & \overline{\mathbf{q}} \\ \underline{\dot{\mathbf{q}}} & \leq & \dot{\mathbf{q}} & \leq & \overline{\dot{\mathbf{q}}} \\ \underline{\boldsymbol{\tau}} & \leq & \tau & \leq & \overline{\boldsymbol{\tau}} \end{array} \tag{3.42}$$

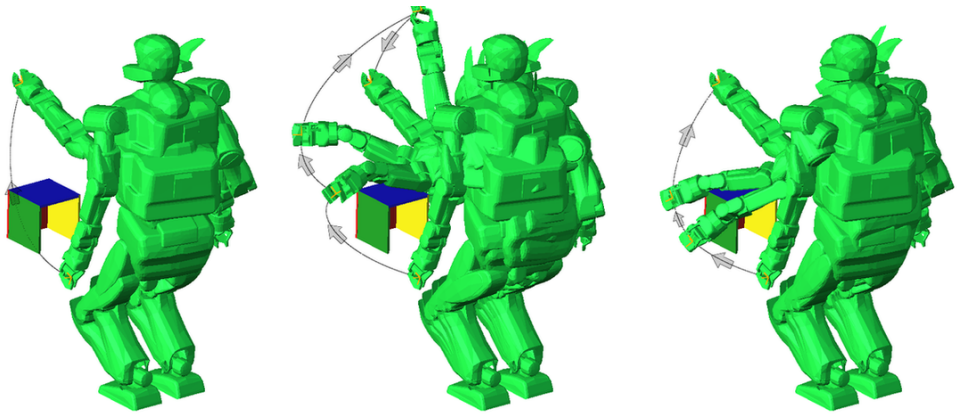
Dynamic balance The robot is submitted in our case to multiple coplanar contact reaction forces from the ground. We can then express the dynamic balance constraint using the whole-body ZMP, which has to remain inside the robot support polygon defined by its feet.

These constraints can be written as for any $t \in [0, T]$:

$$\begin{array}{rcl} \mathbf{p}_{lf}(\mathbf{q}(t)) & = & \mathbf{p_{lf}}(\mathbf{q}(0)) \\ \mathbf{p}_{rf}(\mathbf{q}(t)) & = & \mathbf{p_{rf}}(\mathbf{q}(0)) \\ \mathbf{p}_{zmp}(\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)) & \in & \mathcal{P}_{sup}, \end{array} \tag{3.43}$$

where $\mathbf{p}_{lf}$, $\mathbf{p}_{rf}$ are respectively the 6D positions of the left and right foot, $\mathbf{p}_{zmp}$ and $\mathcal{P}_{sup}$ are the 2D ZMP coordinates and the support polygon respectively. Note that in order to ensure $\mathbf{p}_{lf}$ and $\mathbf{p}_{rf}$ are continuous functions, we do not use Euler angles (such as roll, pitch and yaw) to represent 3D rotations, and each contact body position is written as the concatenation of a 3D translation vector and a 3D rotation vector; the rotation vector direction gives the rotation axis, and its norm gives the rotation angle around this axis.

Collision avoidance constraints We use the capsule-capsule and capsule-environment pairs defined in Section 3.8. Given a configuration $\mathbf{q}$ of the robot, we check that distances for pairs of bodies and pairs of body and obstacle are positive to ensure (self-)collision avoidance. We first tried to add one constraint per pair,



(a) Initial invalid path. (b) RRT path. (c) Shortcut path.

Figure 3.10: Paths for the test case.

which added up to $(327+23)*n_{ms}$ constraints, where n_{ms} is the number of shooting nodes in MUSCOD-II. This led to poor performance as the solver systematically went beyond the threshold number of iterations. We hence propose to define one inequality constraint per robot body, with its value being the minimum distance for all distance pairs involving this body, i.e. both self-collision and obstacle collision pairs.

3.10 Results

We demonstrate the effectiveness of our optimal motion planning framework by first using it in a a simple test case example, then applying it to generate feasible motions on the robot HRP-2. All tests were run on a computer with a 2.53 GHz Intel® Core™2 Duo processor.

3.10.1 Test Case

Figure 3.10a shows the motion planning problem to be solved: HRP-2 starts from its rest position and moves to a goal configuration by raising its left arm. A concave object is placed such that the left hand is at one point enclosed in it if the initial path connecting the start to goal configuration is executed. This is a typical example of a problem with a local minimum defined by the environment, where a real-time control approach in task-space might fail. Figure 3.10b shows a possible solution path found with constrained RRT. This path can be shortened with a shortcut optimizer, as in figure 3.10c.

To showcase the usefulness of our approach, we try to solve the optimal control problem defined in Section 3.9.3 starting from the different paths, and put all results in Table 3.2. When starting with the initial path from figure 3.10a, the solver failed to achieve a single iteration. This can be explained by the fact that in the middle

Table 3.2: Test Case Computation Times

Initial guess	Initial path	RRT path	Shortcut path
Planning time (s)	–	5	5
Shortcut time (s)	–	–	4
Optimization status	ERROR	MAX_ITER	OK
SQP iterations	–	200	70
Optimization time (s)	–	3068	1186
Constraints evaluation time (s)	–	2176	847

of this path, the robot left hand is enclosed inside the obstacle and some distance constraints are violated; the solver fails to determine a clear direction which would remove this violation due to the geometric local minimum. Since the constrained RRT avoids it and generates a collision-free path, the solver behaves correctly when starting with the path in 3.10b, but the maximum number of iterations is reached before reaching convergence. It is achieved when starting with the shortcut path in 3.10c.

In Figure 3.11, the evolution of the same component for the generalized position, velocity, acceleration and jerk is shown. We can see that the state trajectory is continuous, smooth, and the intermediate stops that were present in the initial guess have disappeared.

Figure 3.12 shows the evolution of all 12 components of the kinematic equality constraints on both feet. As the constraints are only enforced on the shooting nodes, we observe slight deviations on the sub-intervals; they are luckily sufficiently small to consider these constraints as perfectly enforced.

Note that about 70% of the optimization time is spent in evaluating the distance constraints and their gradients; this significant overhead can be explained by the fact that MUSCOD-II relies on internal numerical differentiation to compute Jacobians. Figure 3.13 shows the evaluation of the collision avoidance constraints: due to the time discretization, one constraint is violated during less than 100ms. This violation does not however exceed 1mm, and this is considered as acceptable as all distance constraints are computed with bounding capsule geometries which already define conservative volumes around the exact geometries.

Finally, we observe in Figure 3.14 that dynamic balance is guaranteed during the whole motion, as the ZMP remains fully inside the support polygon.

3.10.2 Dynamic Motion Generation on the HRP-2

We also use our approach to generate fast optimal collision-free trajectories and execute them on the humanoid robot HRP-2. In the first scenario, HRP-2 executes a kind of martial art figure where it crosses its arms rapidly while bending its knees,

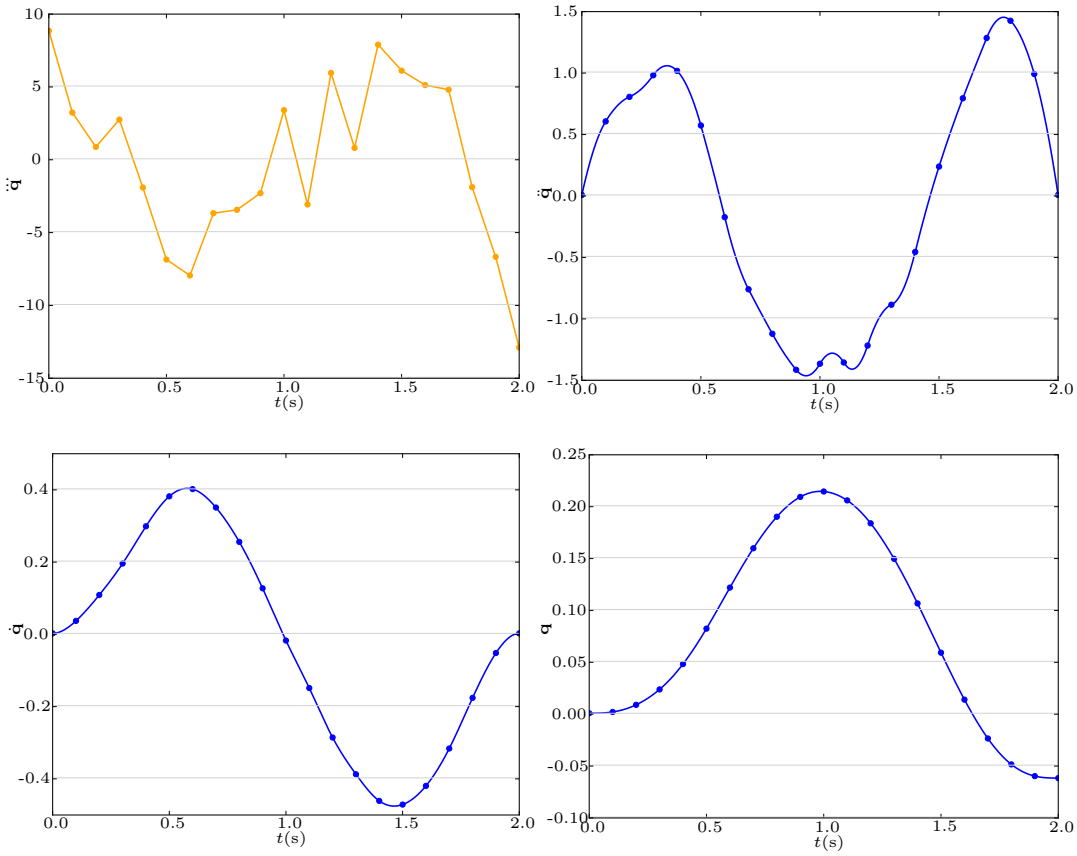


Figure 3.11: Test case: optimized position, velocity, acceleration and jerk trajectories for the chest yaw joint. Note that compared to Figure 3.9, there are no more intermediate stops and the trajectories are smooth.

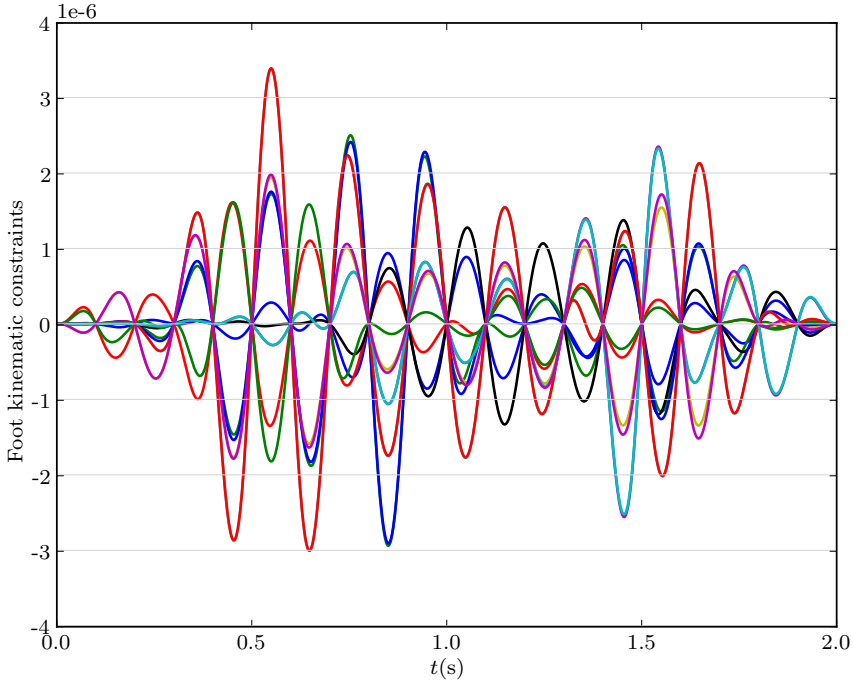


Figure 3.12: Test case: evolution of kinematic constraint values over time. As we constrain the 6D positions of the two robot feet, we end up with 12 equality constraints. They are only fully enforced on the shooting nodes, but it can be seen that the maximum deviation does not exceed a few μm.

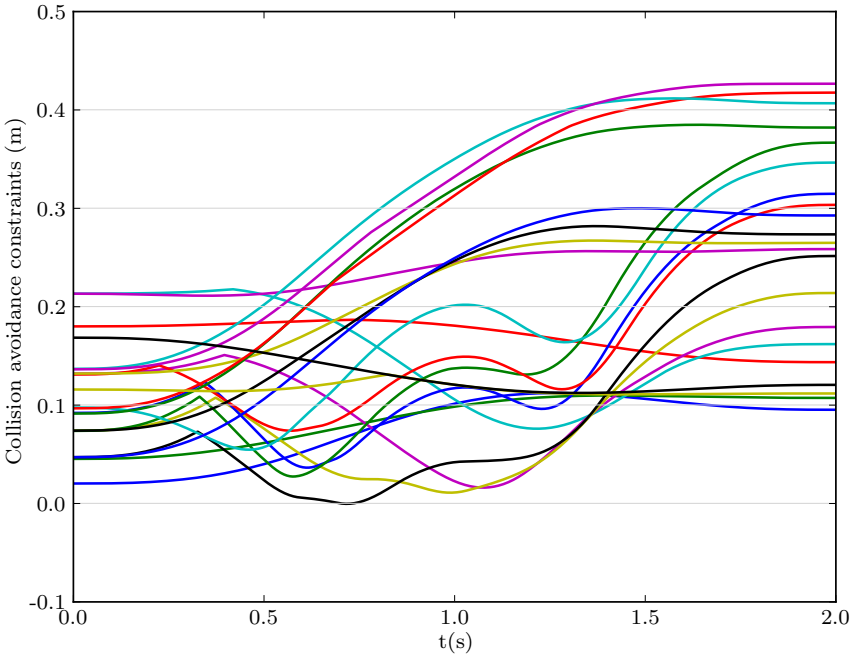


Figure 3.13: Test case: Evolution of the distance inequality constraint values over time. As the constraints are only enforced at the shooting nodes, a slight penetration of a maximum of 1mm is observed between the left forearm capsule and the obstacle.

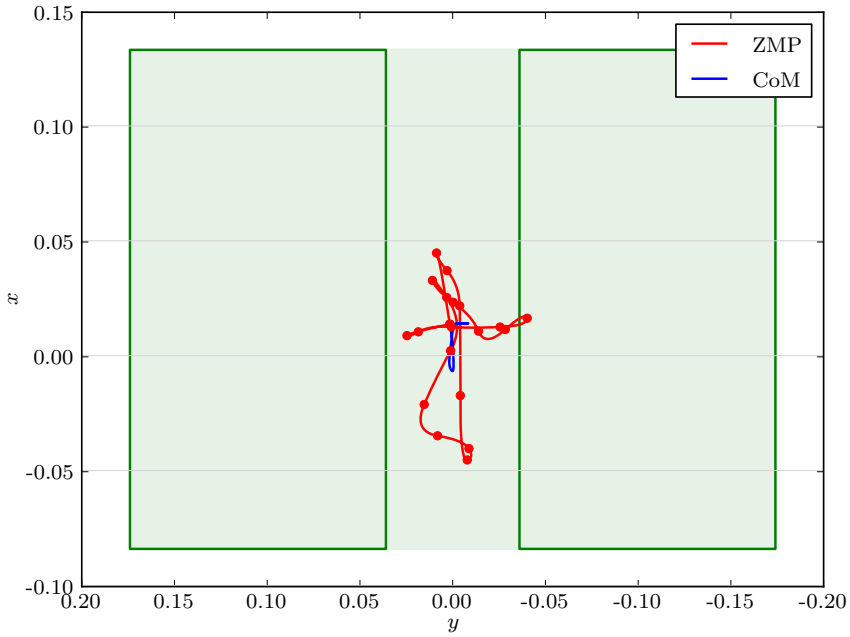


Figure 3.14: Test case: Trajectory of the ZMP and the CoM vertical projection on the floor. The green rectangles correspond to the robot left and right foot, and the filled area is the support polygon. As the ZMP never goes out the support polygon, this trajectory is dynamically balanced.

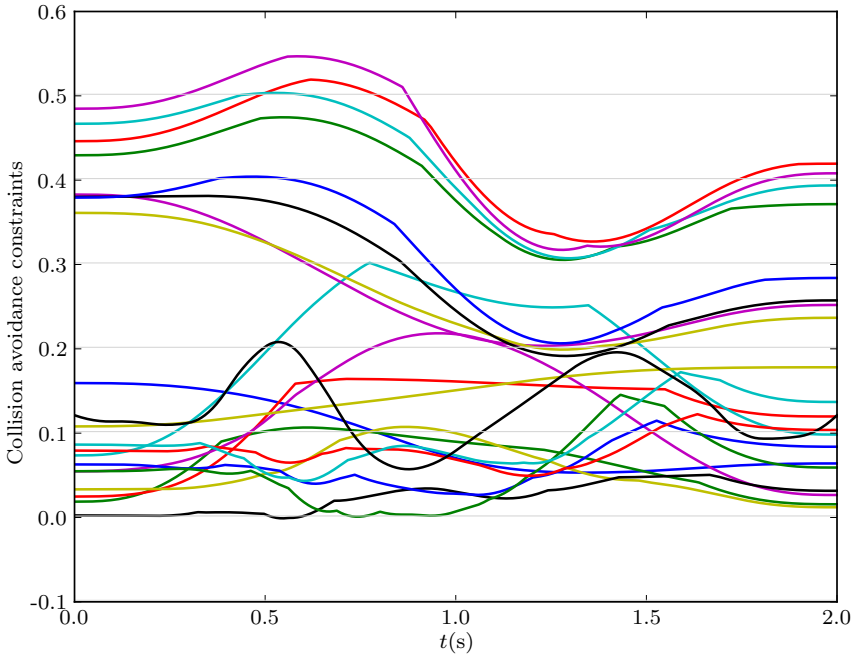


Figure 3.15: Martial arts scenario, phase 2: evolution of the distance inequality constraint values over time. A slight penetration of a maximum of 3mm is observed between the left and right forearm capsules.

changes the arms configuration, then moves back to a rest posture. The motion must be executed while ensuring the arms do not collide with each other, and the robot does not fall. This is quite a difficult task as the 3 trajectories durations are fixed to 1, 2, and 2 seconds respectively. Particularly, the second motion where one arm goes around the other arm proved to be impossible to generate without a prior planning phase as proposed in our approach.

In the second scenario, we add a complex environment that contains shelves with different levels; HRP-2 first bends its knees to grab a ball located deep on the lower shelf, then moves it to an upper shelf to release it between two other objects. The trajectories last respectively 2 and 5 seconds. Here, both collision and self-collision constraints need to be enforced in order to obtain a valid trajectory. Again, the ball transfer motion cannot be generated using an optimal control solver and a simple initial guess; a prior planning phase is needed to find a collision-free transfer path.

We successfully apply our framework to generate feasible motions for both scenarios as seen in figures 3.19 and 3.20. Computation times are shown in Tables 3.3 and 3.4. Figures 3.15 and 3.18 show the collision avoidance constraint evolution for the second phase of each scenario. The ZMP is plotted in Figure 3.16 for the martial arts scenario. In the shelves scenario, Figure 3.17, left, shows that the ZMP inequality constraint which corresponds to the front edge of the feet is active at three nodes; as this constraint is not necessarily enforced on the sub-intervals, it is

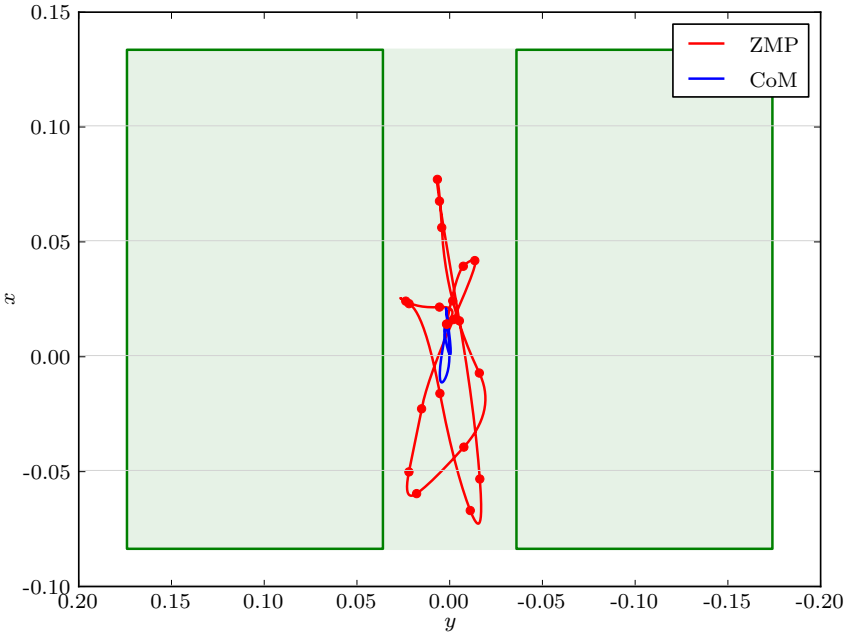


Figure 3.16: Martial arts scenario, phase 2: trajectory of the ZMP and the CoM vertical projection on the floor. The green rectangles correspond to the robot left and right foot, and the filled area is the support polygon. As the ZMP never goes out the support polygon, this trajectory is dynamically balanced.

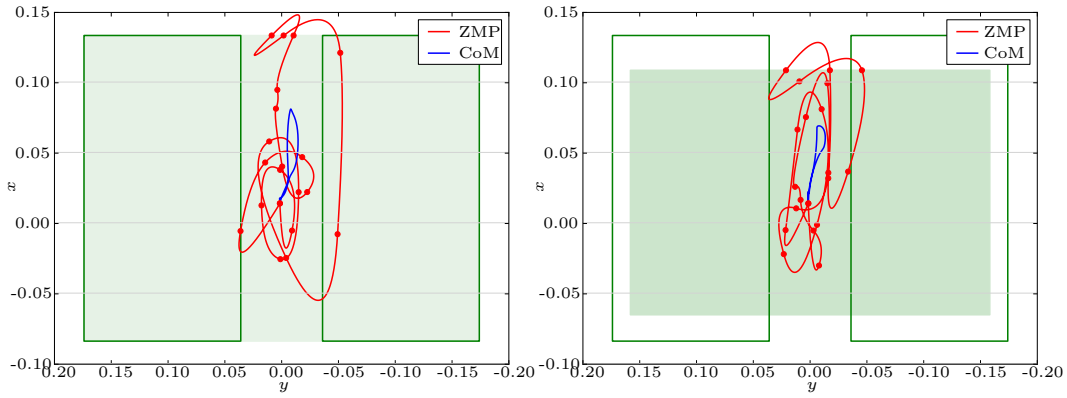


Figure 3.17: Shelves scenario, phase 2: trajectories of the ZMP and the CoM vertical projection on the floor. Left: the dynamic balance constraints are enforced at the nodes, but the ZMP goes out the support polygon in two sub-intervals. Right: the ZMP stays inside the support polygon when more restrictive balance constraints and, leading to a reduced support polygon, are used.

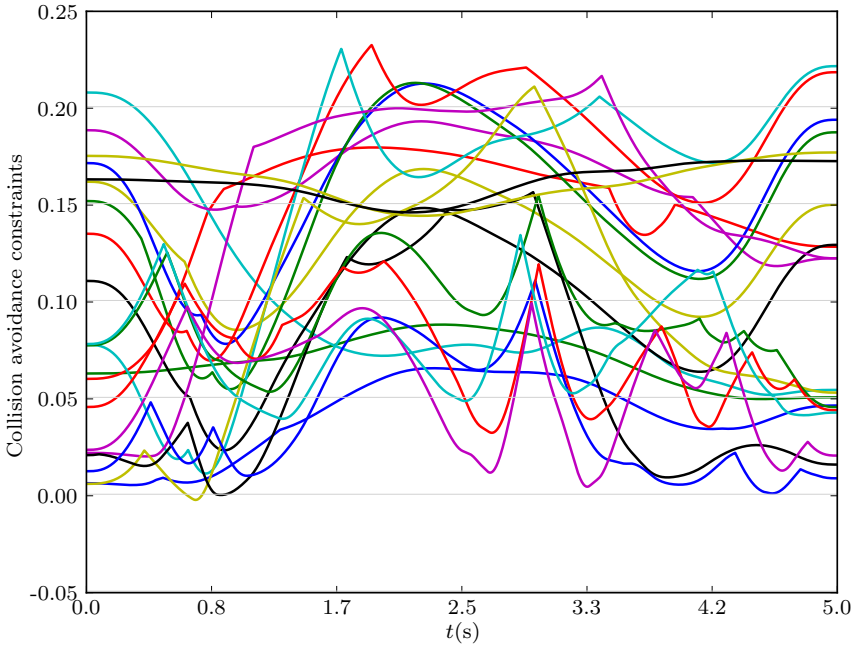


Figure 3.18: Shelves scenario, phase 2: evolution of the distance inequality constraint values over time. A slight penetration of a maximum of 3mm is observed between the right gripper capsule and the shelves.

violated, and this leads to a fall of the robot. One solution would be to increase the trajectory execution time so that the ZMP is closer to the CoM projection. As we still want to generate fast motions, it is necessary to implement balance constraints with a reduced support polygon to ensure that the ZMP remains entirely inside the convex hull of the foot contact points, even inside the time sub-intervals, see Figure 3.17, right.

Table 3.3: Computation Times for Martial Arts Scenario

Phase	1	2	3
Planning time (s)	4	13	2
Shortcut time (s)	4	6	1
SQP iterations	32	73	25
Optimization time (s)	346	1130	278
Constraints evaluation time (s)	124	356	83

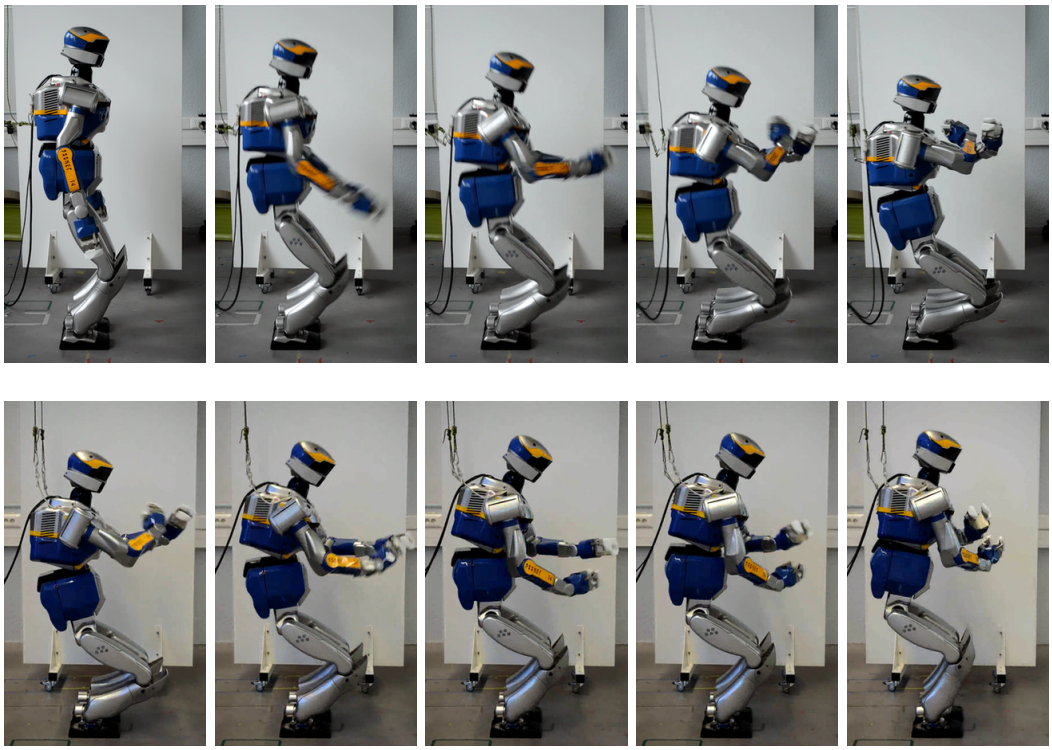


Figure 3.19: HRP-2 does a quick martial arts motion while avoiding self-collision.

Table 3.4: Computation Times for the Shelves Scenario

Phase	1	2
Planning time (s)	13	38
Shortcut time (s)	6	23
SQP iterations	74	80
Optimization time (s)	1745	5020
Constraints evaluation time (s)	1396	2640

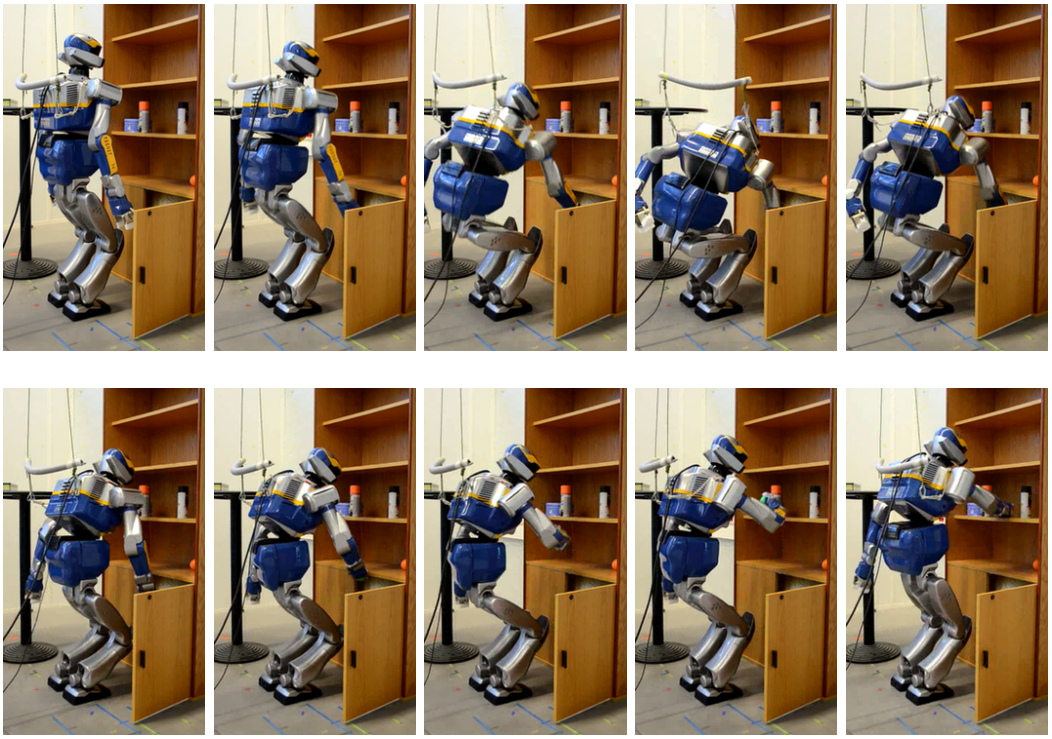


Figure 3.20: HRP-2 bends down quickly to grab a ball in the lower shelf and transfers it to the upper shelf.

3.11 Extension to Non-Coplanar Contact Points

While the previous OCP allows us to successfully generate optimal collision-free trajectories, it is limited to the case where all contact points lie on the same plane, which allows us to ensure dynamic balance through ZMP constraints. We propose to extend our framework in order to handle dynamic balance with potentially non-coplanar contact points and provide some preliminary results.

Therefore, we modify our formulation a bit by keeping the state vector and adding to the control vector all 3D contact forces which are applied on the robot bodies:

$$\begin{aligned} \mathbf{x}(t) &= [\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)]^T \\ \mathbf{u}(t) &= [\dddot{\mathbf{q}}(t), \mathbf{f}_c(t)]^T, \end{aligned} \tag{3.44}$$

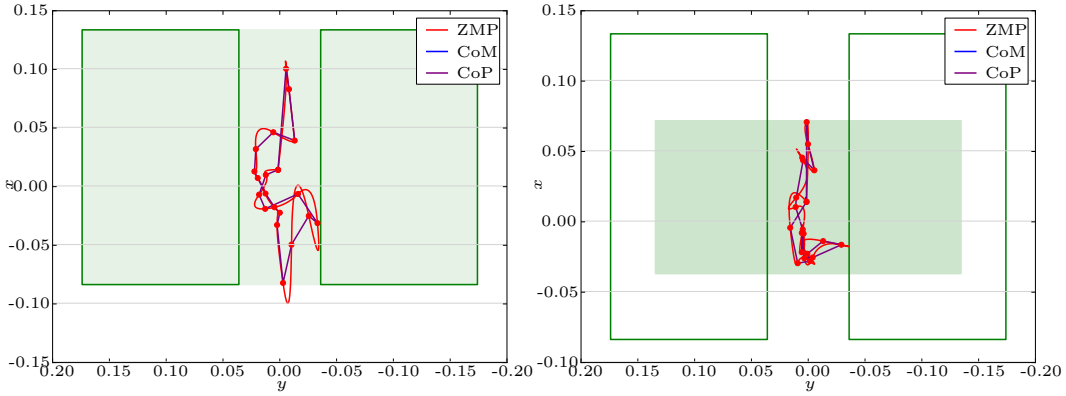
and discretize the forces to represent them as continuous piecewise linear function, as for jerks. Each 3D force vector is expressed in the local body coordinate system.

As the ZMP is no longer useful in this case, we replace the ZMP inequality constraints by the dynamic balance equality constraint from Equation (3.33). For now we assume that the limiting coefficient of static friction μ_s is infinite, i.e. that the friction cone is the positive half-space. We therefore only add the first constraint from (3.28), which ensure that the contact force normals are positive.

We use the second phase of the martial arts scenario to verify our approach with a set of coplanar contact points. As both feet of the HRP-2 are on the horizontal floor, we first place one contact force on each of the four foot vertices, which amounts to a total of 8 forces. We set the motion duration to 0.6s in order to create a fast trajectory.

Let the *center of pressure (CoP)* be the contact point barycenter, weighted by the normal contact forces. We plot the CoM, ZMP and CoP in Figure 3.21a. Interestingly, the ZMP and the CoP coincide over the shooting nodes, and differ only in between, which confirms that the new dynamic balance formulation is equivalent to the one using the ZMP when all contact points lie in the same plane. We also notice that the ZMP is not always inside the support polygon as the dynamic balance constraints are not enforced over the sub-intervals. Figure 3.21b shows the resulting ZMP and CoP trajectories when placing contact forces on a reduced support polygon vertices.

We show in Figure 3.22 the normal contact force values over time. Note that as the forces are required to be non-vanishing, their normal components remain positive during the whole motion. We also show the floating joint generalized force constraint value in Figure 3.23; as expected, all components are equal to zero at the shooting nodes, but the constraint are not properly enforced in the sub-intervals. The values may seem high, but they should be compared with other forces to which the robot is subject, such as the gravity force. For instance the red and blue line correspond to linear force components, and they represent less than 5% of the HRP2 weight, which is around 550N.



(a) Contact forces applied at the foot edge vertices.

(b) Contact forces applied on reduced support polygon.

Figure 3.21: Martial arts scenario, phase 2: trajectories of the ZMP and the CoP computed from the contact forces. Left: the dynamic balance constraints are enforced at the nodes, but the ZMP goes out the support polygon in two sub-intervals. Right: the ZMP stays inside the support polygon when more restrictive balance constraints, leading to a reduced support polygon, are used.

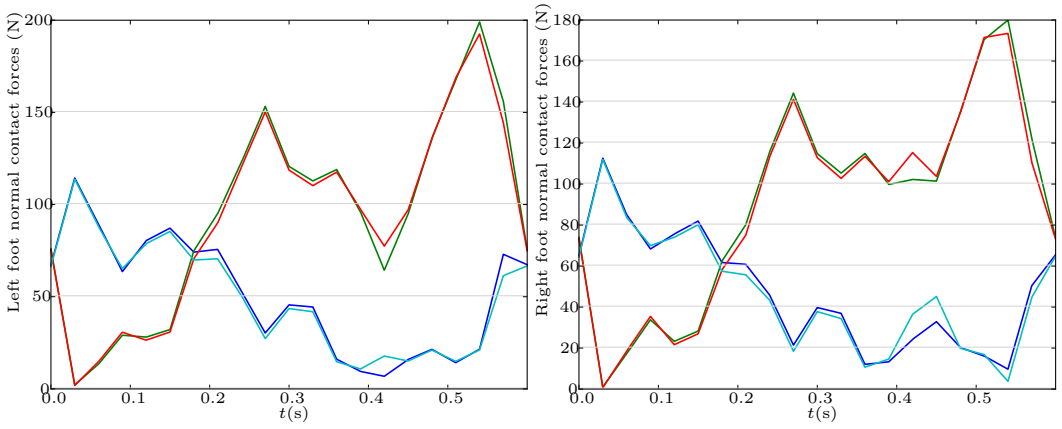


Figure 3.22: Martial arts scenario, phase 2: Left and right normal contact forces. All normal forces remain positive during the motion, and are computed to ensure dynamic balance.

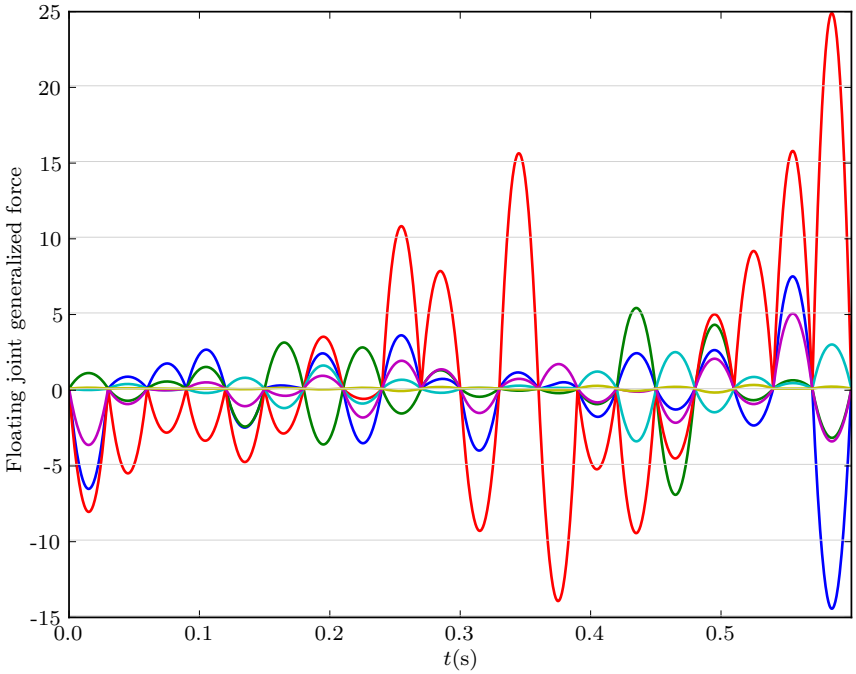


Figure 3.23: Martial arts scenario, phase 2: floating joint generalized torque evolution over time. It is required to be equal to 0 in order to obtain dynamic balance.

Finally, we confirm in Figure 3.24 that the collision avoidance constraints are properly enforced.

In this work, we only tried enforcing dynamic balance constraints with a contact force formulation in the case of horizontal coplanar contact points. The same formulation can luckily be used for cases where the contact points are not coplanar, e.g. where HRP-2 stands on an uneven floor or uses its upper limbs to go in contact with the environment. We leave this for future work.

3.12 Discussions and Future Work

In subsection 3.10.1, we demonstrate in a simple example the influence of the initial guess of the optimal control problem on the solver success and performance. In fact, due to its probabilistic completeness, the usage of the constrained planner in a first stage guarantees that an initial collision-free and quasi-statically feasible trajectory can be found. The optimization solver then can reshape this trajectory in order to minimize the objective function while enforce constraints such as joint limits and dynamic balance.

Solution optimality Note however that with this method, only locally optimal trajectories are found; further investigation can be done in order to try to transform the OCP into an equivalent convex optimization problem, as this would allow

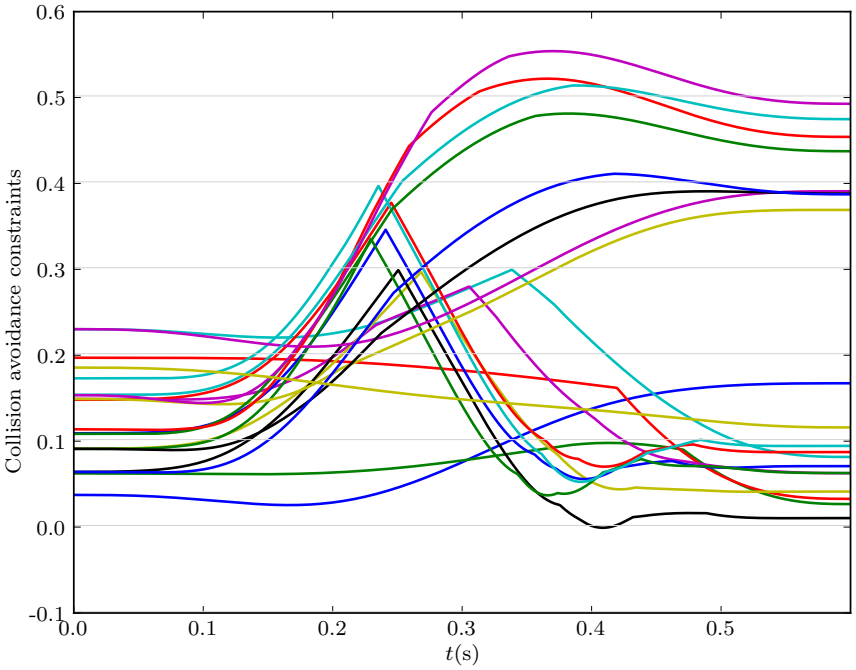


Figure 3.24: Martial arts scenario, phase 2: collision avoidance constraints evolution over time. The constraint violation does not exceed 2mm on the sub-intervals.

both using simpler and more efficient solvers and ensuring that global minimizers are computed. Such an approach has been applied for time-optimal path tracking problem, see [Verscheure 09]. Alternatively, it might be interesting to solve the optimal motion planning problem in one step using the RRT* sampling-based planner, as described in Section 3.3.4.

Trajectory duration In this work, the trajectory duration is fixed in the OCP formulation. This means that if it is not properly set, the optimization solver might fail as some constraints such as velocity limits would never be enforced. This issue could be solved by including time as a variable in the OCP in order to guarantee the complete framework succeeds in generating optimal trajectories. However, this has to be done carefully: indeed, if time is a free variable, the minimum-jerk trajectory will have an infinite duration. One could choose to include time in the objective function, so that both jerk and time are minimized, but a suitable weight ratio for both terms then needs to be found, and the optimal trajectory quality would be directly affected by those weights. An alternative solution would be to add an additional stage in the framework: the initial path would be first parameterized to an infinite-duration trajectory, then a constrained minimum-time OCP would be solved, and the solution would be finally used as an initial guess to solve the constrained fixed-time minimum-jerk OCP we described in this work.

Computation time Obviously the optimal motion planning we described is still not meant for online motion generation. In its current state, the two major hurdles are the distance constraint computation and the optimal control solver itself. We use the KCD library in order to successfully compute distances between capsules and meshes; however, it not the most efficient library and other ones such as the Flexible Collision Library (FCL) [Pan 12], can be used for better performance. Regarding the optimal control solver, MUSCOD-II relies on an internal numerical differentiation mechanism to compute all Jacobians. This is a very nice feature which allowed to formulate an OCP very quickly without explicitly derivating the Jacobians, and it leads to acceptable performance when the functions are not expensive to compute. Sadly computing distances between capsules and meshes is not very fast. Luckily their Jacobian is not very hard to compute; it would be then useful to rely on solvers which can use the Jacobian explicit expression.

One other possibility we did not explore is parallel computing; indeed, dynamics, distance computation, and multiple shooting algorithms all support parallelization; relying on smart GPU-based implementations of these algorithms would obviously lead to enhanced performance.

Enforcing constraints MUSCOD-II is based on a direct multiple shooting method. This implies, as we saw in Section 3.3.3, that the OCP is transcribed to a finite-dimensional NLP thanks to a discretization of both the control and state vector. This also implies that constraints are only enforced on the multiple shooting nodes, without any guarantee that they will be valid over the time sub-intervals. As long as we use enough nodes, this is not really an issue for constraints that have slow dynamics such as kinematic constraints. This is not the case for balance constraints such as the ZMP, and we needed to verify a posteriori that they were enforced over the whole trajectory, reducing the support polygon size when necessary.

In [Lengagne 13], this issue is solved by locally approximating constraints with Taylor polynomials, computing their extremal values over each sub-interval of the time-grid, and using them as effective constraints in the OCP formulation. Another possible solution consist in relying on direct collocation methods; as the control and state are finely discretized, constraints can be verified almost everywhere on the trajectory.

Extension to non-coplanar contacts We showed in Section 3.11 preliminary results of optimal control with a unilateral contact force formulation which ensures balance. While we only applied it to horizontal coplanar contact points, it can be used in the same way to generate optimal motions with non-coplanar contact points. If we want to handle correctly such cases in our optimal motion planning framework, we equally need to plan statically balanced paths for non-coplanar contact points. This implies redefining the planning manifold $\mathcal{M}$ in order to enforce a generic stability criterion, as described in [Bretl 06], where a configuration and contact forces are computed such that static balance is obtained.

Towards collision-free locomotion optimization In this work, we focused on generating motions under the assumption that the set of contact points does not change during the whole motion. It would be however interesting to consider optimal motion planning for the more general case of locomotion, manipulation, or both. As in [Lengagne 13], our framework could be extended, by planning and optimizing sequentially for multiple phases, where each phase is defined by a fixed set of non-coplanar contact points. This approach requires us to first plan the contact stance sequence which cannot be changed later on.

An alternative solution would be to reformulate the OCP so that changing contact point sets are taken into account, e.g. through a linear complementarity constraint between a contact point distance to obstacles and the normal contact force, as proposed in [Posa 12, Tassa 12, Mordatch 12]. Collision avoidance can be integrated in this process by first planning a draft path, using high-level planners such as the one we described in Chapter 2, and letting the optimizer decide which contact points to use, and when to use them.

3.13 Conclusion

In this chapter we propose a novel approach to tackle optimal control problems in cluttered environments. Our approach combines, in a two-stage framework, a constrained path planning algorithm and an optimal control problem solver. We generate optimal feasible trajectories for the humanoid robot HRP-2 and successfully execute them in constrained environments.

Our framework can benefit from improvements to increase its usability. In future work we aim to consider non-coplanar contacts, as well as release the robot from its fixed support constraints in order to accomplish optimal locomotion planning.

Chapter 4

Conclusion

4.1 General Contributions

The work presented in this thesis deals with planning optimal motions for anthropomorphic systems in general, and humanoid robots in particular. Based on promising but still separate advances in motion planning and optimization in high-dimensional spaces, the main focus of this work was set on the combination of methods from both fields in order to generate optimal collision-free trajectories for humanoid robots.

More precisely, the contributions lead to the development of :

- an efficient A*-based path optimization algorithm for a bounding-box representation of a humanoid robot. It was inserted in an existing two-stage planner which allowed the generation of minimum-time dynamic walking trajectories.
- a whole-body motion planner for humanoid robots. Based on the small-space controllability property which was established in this work, collision-free trajectories that seamlessly combine dynamic walking and manipulation were produced.
- a two-stage optimal motion planning framework which combines constrained sampling-based planners with optimal control methods. Dynamically-feasible collision-free smooth trajectories were successfully generated in constrained environments.

Finally, special care was given in this thesis to generate feasible motions that not only can be executed on digital actors, but also on physical humanoid robots in real environments. Therefore, resulting motions from all previously cited contributions were successfully executed on the HRP-2 humanoid robot.

4.2 Perspectives

Let us recall that planning represents only one of the three components of the perception-planning-action paradigm. We want humanoid robots to achieve tasks

such as locomotion in a reactive way; it is therefore obvious that the time which is spent in the planning component should be very short, i.e. motions should be generated very quickly. This work succeeded in the development of generic methods for planning optimal motion for humanoid robots, but their computational efficiency remains one of their main limitations, as computation time was not the focus of this work. Some pointers were thus given in the previous chapter discussions in order to address this issue.

Humanoid robots have formidable abilities when compared to more common fixed-base or wheeled robots: thanks to their legged structure, they can walk, step over obstacles, run, climb hills and execute acrobatic figures among many things. This thesis was limited to generating motions where a humanoid robot is standing on a flat horizontal floor; this voluntary limitation allowed us to devise sound methods for whole-body dynamic walking and optimal motion planning. Nevertheless, insights for extending our optimal motion planning framework to handle a set of non-coplanar contact points were given.

In some sense, both the computer graphics community and the robotics one share the common goal of generating feasible motions, with the former's main concern being the motion realism with respect to the laws of Physics, and the latter's being its feasibility on complex hardware platforms in the physical world. Thanks to a smart usage of online model-predictive control techniques for digital avatar animation [Coros 10, Tassa 12], recent results have shown that we can hope to make humanoid robots plan extreme locomotion reactively. However, beside a transcription of the above algorithms to humanoid robots, we need to make sure that we have both the right hardware and control software.

Consider for instance a back-flip motion, as shown in Figure 4.1. The robot must first exert adequate forces on the floor to jump in the air, rotate during the flying phase, then land back on its feet. Most mainstream humanoid robots, such as the HRP-2, are capable of pushing sufficiently to jump in the air, but their actuators are such that they cannot quickly transform the kinetic energy acquired in the flying phase into other energies when they hit the ground; this energy is thus entirely transmitted to the mechanical structure and might cause it to break. Introducing backdrivable actuators in the robot structure can help addressing this issue by allowing the fast transformation of the kinetic energy back into electric energy and keeping the physical system integrity. Furthermore, the generated current measurements can give a good idea of the actuator torques, which paves the way for the implementation of force-based control laws. Such laws, compared with position-based control laws, can be extremely useful in retaining the robot balance during fast motions.

In light of these results, we would like to tackle, in our future works, the themes of model-predictive control and stabilization for force-controlled humanoid robots.

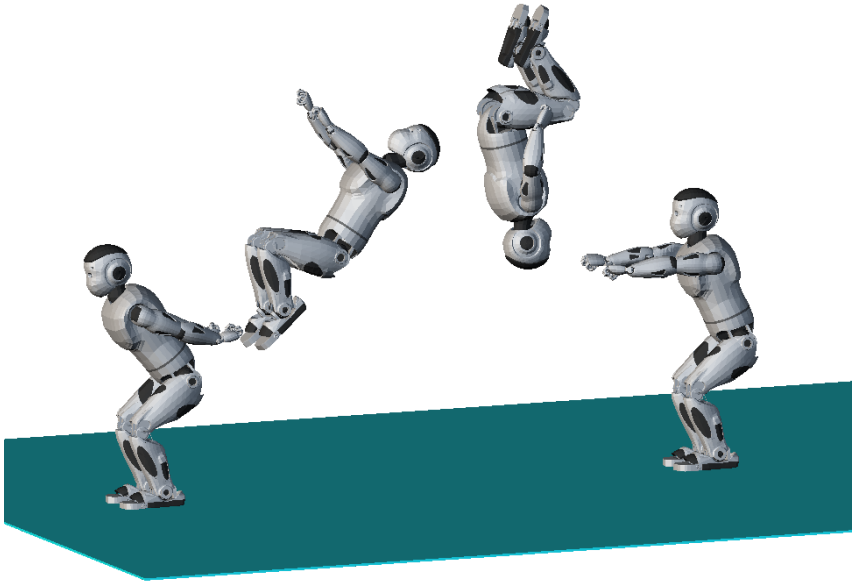


Figure 4.1: A humanoid robot executes a back-flip. This is a highly dynamic motion where the flying phase is difficult to control, and the foot forces on impact may be high.

Appendix A

Sliding Motion Planning Benchmarks

The motion planning algorithms presented in Chapter 2 have been implemented using KineoWorks^{TM}[Laumond 06]. The planning times have been measured on an Intel Core 2 Duo 2.13 GHz PC with 2 GB of RAM. Evaluation of the randomized algorithm has been conducted by executing 500 trials on each scenario using two flavors of RRT: the classic RRT and IPP-RRT [Ferre 04]. We present the results in Figures A.1, A.2 and A.3.

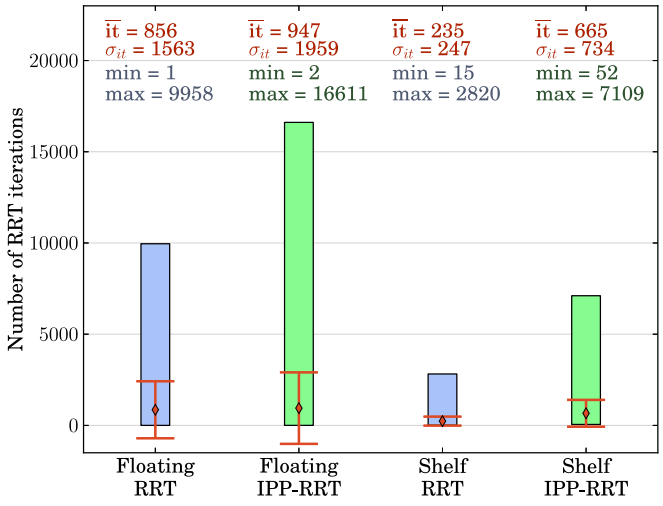


Figure A.1: Number of RRT iterations it for the floating objects and the shelf scenarios, using two variants of RRT. Mean $\overline{it}$, standard deviation σ_{it}, minimum and maximum values are represented.

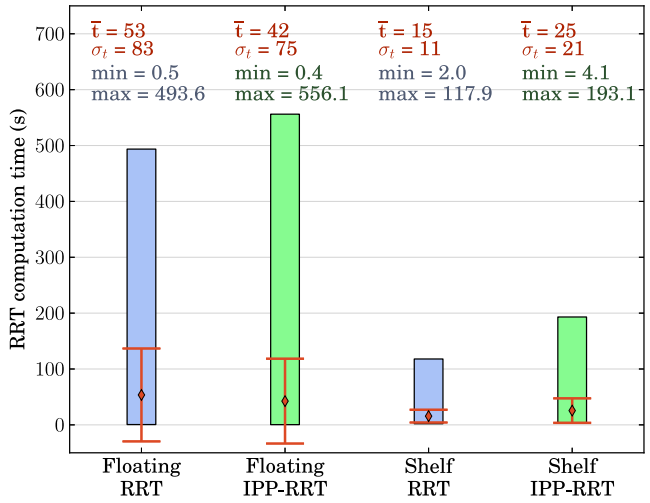


Figure A.2: RRT computation time t for the floating objects and the shelf scenarios, using two variants of RRT. Mean $\bar{t}$, standard deviation σ_t, minimum and maximum values are represented.

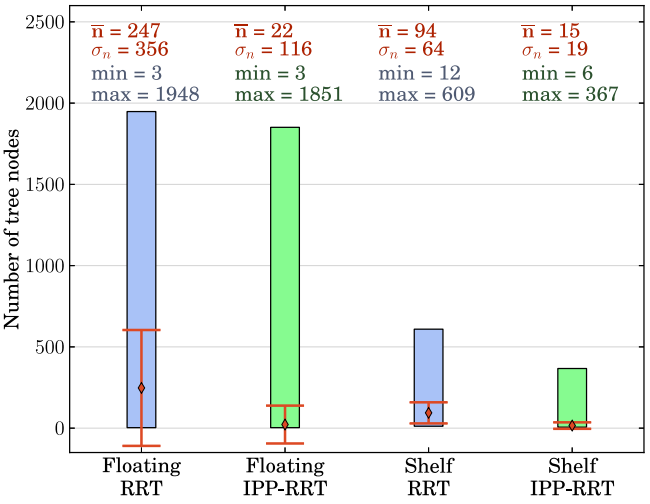


Figure A.3: Number of tree nodes n for the floating objects and the shelf scenarios, using two variants of RRT. Mean $\overline{n}$, standard deviation σ_n, minimum and maximum values are represented.

Appendix B

Numerical Optimization

We give here an overview of the most successful numerical optimization techniques that can be found in the literature. We focus on Jacobian-based methods, i.e. methods that use information given by the *variations* of the function we want to minimize to find its minimizer. As this section is largely based on [Nocedal 99], we invite the interested reader to refer to it for more details.

B.1 Unconstrained Optimization

Given a scalar *objective function* $f : \mathbb{R}^n \to \mathbb{R}$, we would like to solve the following problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{B.1}$$

This is a problem of *unconstrained optimization*; it consists of finding one ore more solutions $\mathbf{x}^\star$, which we call *minimizers*. A minimizer is said to be *global* if and only if:

$$f(\mathbf{x}^\star) \leq f(\mathbf{x}) \ \forall \mathbf{x} \in \mathbb{R}^n, \tag{B.2}$$

implying that there is no other point $\mathbf{x} \in \mathbb{R}^n$ such that the value of f at $\mathbf{x}$ is lower than the value of f at $\mathbf{x}^\star$. On the other hand, a *local minimizer* is such that $\exists$ a neighborhood $\mathcal{N}$ of $\mathbf{x}^\star$ and:

$$f(\mathbf{x}^\star) \leq f(\mathbf{x}) \ \forall \mathbf{x} \in \mathcal{N}. \tag{B.3}$$

Obviously, a local minimizer is weaker than a global minimizer, as Equation (B.3) implies that there is no other minimizer only in the vicinity of $\mathbf{x}^\star$, and it does not ensure the nonexistence of another point $\mathbf{x}_g^\star$ such that $f(\mathbf{x}_g^\star) \leq f(\mathbf{x}^\star)$. Therefore, there is no guarantee that the local minimizer is also a global one for f.

B.1.1 Necessary Conditions

In order to characterize a minimizer of f, we introduce the *first-order necessary conditions* for unconstrained optimization:

Theorem 3. *$\mathbf{x}^\star$ is a local minimizer, f continuously differentiable (C^1) in an open neighborhood of $\mathbf{x}^\star \Rightarrow \nabla f(\mathbf{x}^\star) = \mathbf{0}$.*

All points $\mathbf{x}^\star$ which satisfy the first-order conditions are called stationary points. Note that stationary points can correspond to minimizers, maximizers, or saddle points of f. The *second-order necessary conditions* are then used to distinguish local minimizers:

Theorem 4. *$\mathbf{x}^\star$ is a local minimizer, f twice continuously differentiable (C^2) in an open neighborhood of $\mathbf{x}^\star \Rightarrow \nabla f(\mathbf{x}^\star) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^\star)$ is positive semi-definite (psd).*

The key for the conditions is that f be at least C^2 so that ∇f and $\nabla^2 f$ be defined and continuous. Note that if f is convex and C^1, $\nabla^2 f(x)$ is positive definite (pd) $\forall\ \mathbf{x}$, and it can be shown that any stationary point of f is also a global minimizer of f.

B.1.2 Finding the Minimizer

One way of finding a minimizer of f is to look for a stationary point starting from an initial point $\mathbf{x}_0$, and produce a sequence of iterates $\{\mathbf{x}_k\}_{k=0}^{\infty}$ that terminates when a termination condition is reached; this condition corresponds to the point $\mathbf{x}^\star$ where no more progress can be made, up to a specified tolerance $\epsilon > 0$.

Given an iterate $\mathbf{x_k}$, the next iterate $\mathbf{x_{k+1}}$ can be chosen based on information about f at $\mathbf{x_k}$ so that $f(\mathbf{x_{k+1}}) < f(\mathbf{x_k})$. There exist two main strategies to do so:

1. Line search strategy: a direction $\mathbf{p_k}$ is first chosen, then a step length $\alpha_k > 0$ is computed such that it approximately solves the 1D minimization problem: $\min_{\alpha>0} f(\mathbf{x_k} + \alpha_k \mathbf{p}_k)$. This leads to $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p_k}$.
2. Trust region strategy: a local model m_k of f is constructed, and a direction p_k is chosen inside a fixed-size trust region such that it approximately solves the minimization problem: $\min_{\mathbf{p_k}} m_k(\mathbf{x}_k + \mathbf{p}_k)$.

Note that these two strategies are very similar as they try to find approximate solutions to simple optimization problems which offer good performance. They mainly differ in the order in which the step length and search direction are chosen. In the following section, we describe some of the most common line search strategies. We illustrate them using the Rosenbrock function [Rosenbrock 60], or banana function, defined as:

$$\mathbf{x} = (x, y) \in \mathbb{R}^2, \quad R(\mathbf{x}) = (1 - x^2) + 100(y - x^2)^2. \tag{B.4}$$

R has only one minimizer $\mathbf{x}^\star = (1, 1)$, and is commonly used to benchmark optimization algorithms.

B.1.3 Steepest Descent Line Search

Let f_k denote $f(\mathbf{x}_k)\ \forall\ k > 0$. Using a first-order Taylor approximation of f, it can be proved that:

$$\mathbf{p_k} = -\nabla f_k \tag{B.5}$$

is the steepest descent direction, i.e. it is the direction along which f decreases the most locally around $\mathbf{x}_k$.

This strategy has a *linear convergence rate*:

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|}{\|\mathbf{x}_k - \mathbf{x}^\star\|} \leq r \text{ for all } k \text{ sufficiently large}, \quad r = \frac{\kappa - 1}{\kappa + 1}, \tag{B.6}$$

where κ is the condition number of the Hessian $\nabla^2 f$, i.e. the ratio of the maximum singular value over the minimum singular value of $\nabla^2 f$. It can therefore lead to extremely slow convergence when $\nabla^2 f$ is ill-conditioned and $r \approx 1$.

Figure B.1 shows an example of minimizing the Rosenbrock function. We show the first 20 iterates when using a steepest descent line search and setting α_k to a constant value of 1. The search direction $\mathbf{p_k}$ is always orthogonal to the function contour line, as it is the direction that allows to decrease R quickly. Unfortunately, as the iterates reach the basin, keeping a constant step size leads to a strong oscillation and convergence is not achieved. This motivates the need for a good step size computation method that will ensure a decrease of the function at each iteration.

The Wolfe conditions offer the theoretical means and an easily verifiable way to compute suitable step lengths. They are defined as:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x_k} + c_1 \alpha_k \nabla f_k^\top \mathbf{p_k}), \tag{B.7}$$

$$\nabla f(\mathbf{x_k} + \alpha_k \mathbf{p}_k)^\top \mathbf{p_k} \geq c_2 \nabla f_k^\top \mathbf{p_k}, \tag{B.8}$$

where $0 < c_1 < c_2 < 1$. Equation (B.7) is called the *sufficient decrease* or *Armijo condition*; it rejects too small decreases in f. Equation (B.8) is called the *curvature condition*, and it rejects too negative slopes which might slow down convergence. In practice, c_1 is very small and set to 10^{-4}, while c_2 is set to a value ranging from 0.1 to 0.9.

The Wolfe conditions can be used to write a simple step length computation algorithm, as described in Algorithm 6. The idea is to start with a large value of α_k and decrease it until the Wolfe conditions are satisfied.

Figure B.2 shows the solution to the same problem as in Figure B.1 with the Wolfe conditions enforced at each iteration. The minimizer $\mathbf{x}^\star = (1\ 1)$ is reached after around 5000 iterations. Such a high number is mainly due to an ill-conditioned Hessian and the induced zigzagging behavior which prevents quickly reaching the minimizer, as shown in Figure B.2b.

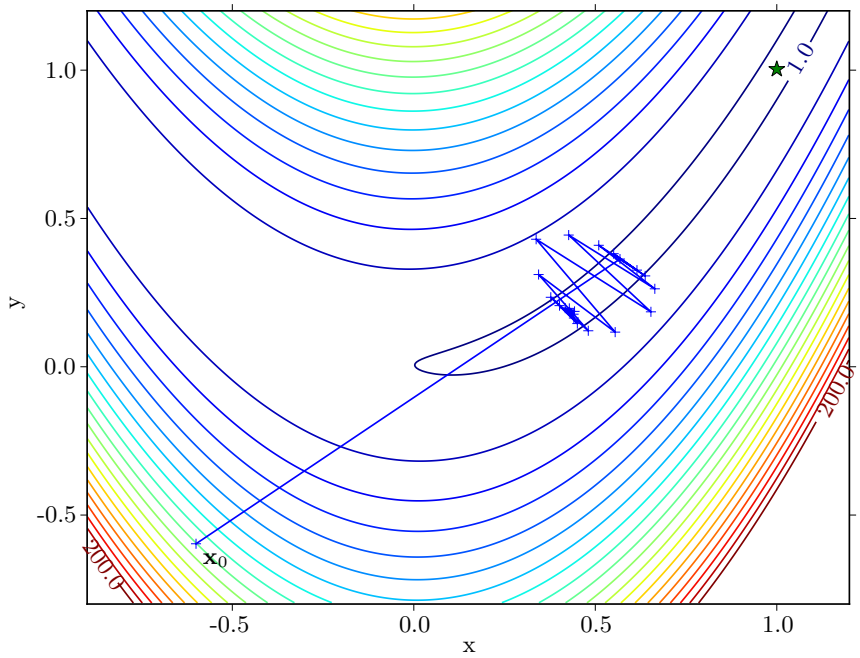
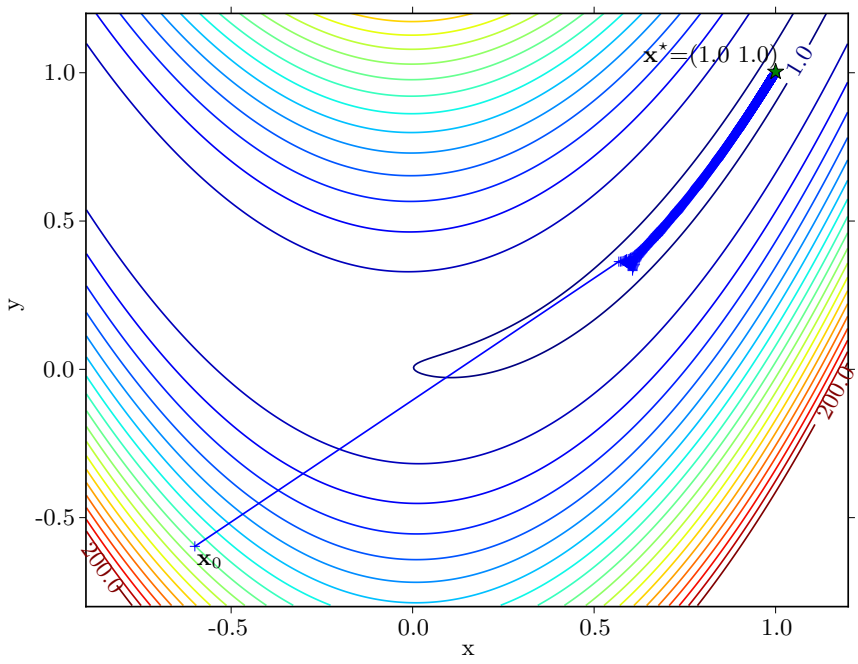


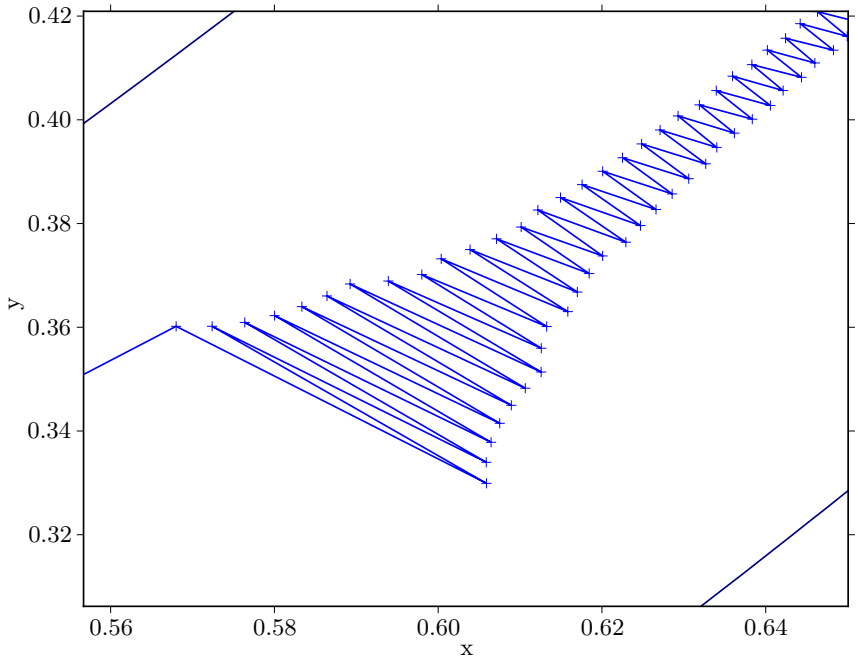
Figure B.1: Contour lines show the Rosenbrock function R. Starting from $\mathbf{x}_0 = (-0.6, -0.6)$, a steepest descent line search is used with a constant step size $\alpha_k = 1$. The minimizer basin is reached very quickly, but large values of the step size then prevent minimizing the function. Note that the steepest descent direction is orthogonal to the contour lines of R.

Algorithm 6 `StepLengthWolfe`(f, $\mathbf{x}_k$, $\mathbf{p}_k$, α_k, ρ, it_max)

$it \leftarrow 0$
while Wolfe conditions are not verified & $it < it\_max$ **do**
  // $\rho < 1$
  $\alpha_k \leftarrow \rho\alpha_k$
  $it \leftarrow it + 1$
end while
return α_k



(a) Solution to the Rosenbrock function minimization problem.



(b) Enlarged view: the zigzagging behavior slows down convergence.

Figure B.2: Steepest-descent line search strategy. The Wolfe conditions are enforced at each iteration to ensure sufficient decrease of the objective function.

B.1.4 Newton Line Search

Assuming that f is C^2 and using a second-order Taylor approximation at $\mathbf{x}_k$, we can derive the *Newton direction*:

$$\mathbf{p_k} = -\nabla^2 f_k^{-1} \nabla f_k; \tag{B.9}$$

$\mathbf{x}_{k+1} = \mathbf{x_k} + \mathbf{p_k}$ is then the minimizer of the local quadratic approximation of f. The Newton direction has a natural step length $\alpha_k = 1$. Nevertheless the Wolfe conditions still need to be verified to make sure that there is a decrease in the exact function f. Figure B.3 shows the sequence of iterates when applying a Newton line search to find the minimizer of the Rosenbrock function. It is clear in Figure B.3b how the next iterate is the minimizer of the local quadratic approximation of the objective function.

The Newton line search is very efficient and exhibits a *quadratic convergence rate*:

$$\frac{\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|}{\|\mathbf{x}_k - \mathbf{x}^\star\|^2} \leq M \text{ for all } k \text{ sufficiently large}, \quad M > 0, \tag{B.10}$$

which is faster than a linear convergence rate. However, beside knowing how to compute the Jacobian of f, we also need to derive the Hessian of f and this is not always a trivial task.

B.1.5 Quasi-Newton Line Search

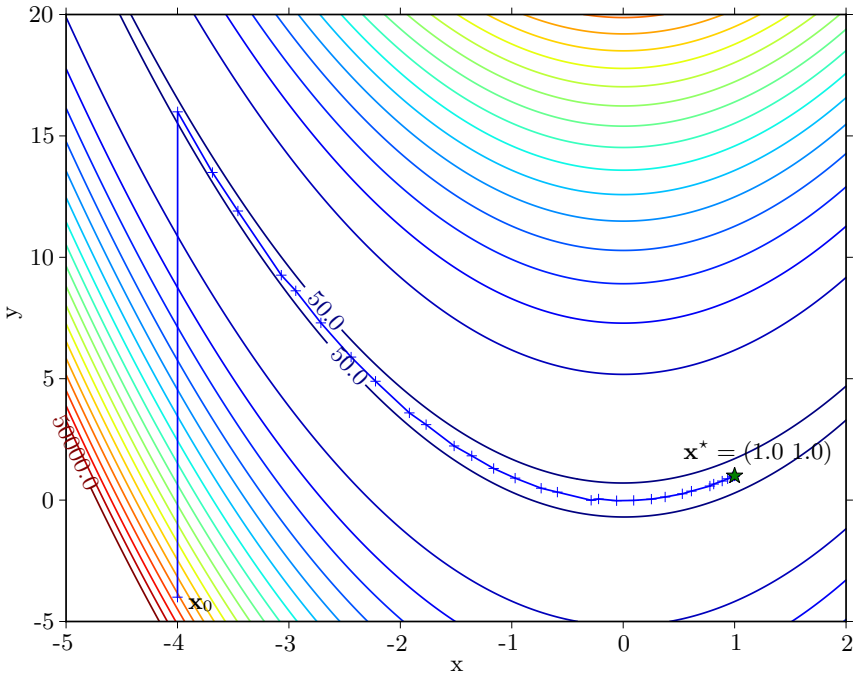
Instead of using the Hessian, a quasi-Newton line search relies on an approximation of $\nabla^2 f$ when it is not possible to compute it. The search direction is then given by:

$$p_k = -\mathbf{B}_k^{-1} \nabla f_k, \tag{B.11}$$

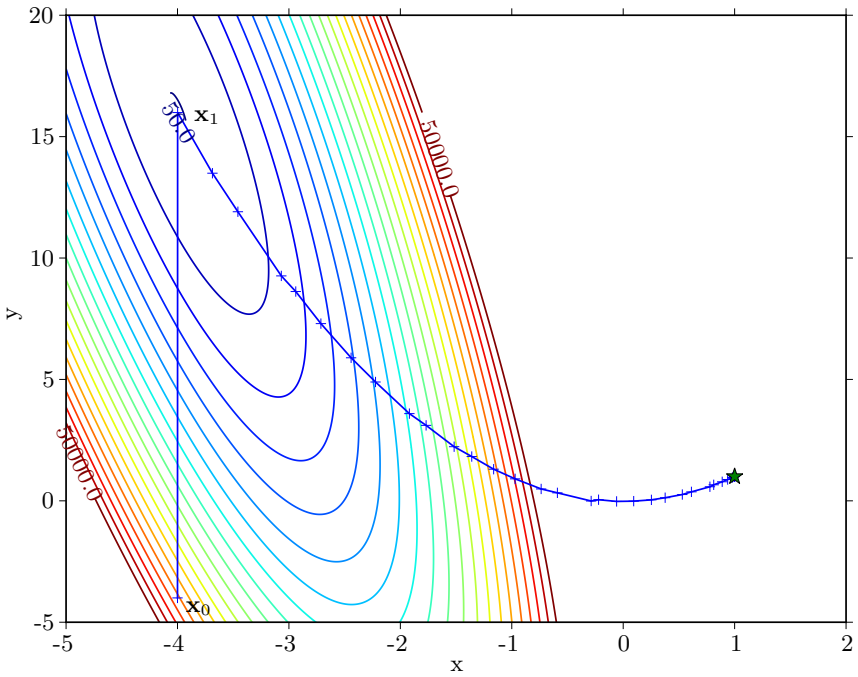
where $\mathbf{B}_k \approx \nabla^2 f$ is a symmetric nonsingular matrix.

Several algorithms for deriving approximates $\mathbf{B_k}$ exist, the best one being the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, as described in Algorithm 7. It relies on updating the $\mathbf{H}_k = \mathbf{B}_k^{-1}$ while the iterative optimization is taking place. As an initial value $\mathbf{H}_0 = \mathbf{I}$ is given, the line search behaves like a steepest-descent line search for the first iterations, converging towards a Newton line search as the optimization advances and as the approximation is closer to the real Hessian. The BFGS line search has then a *superlinear convergence rate*:

$$\lim_{k \to \infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|}{\|\mathbf{x}_k - \mathbf{x}^\star\|} = 0, \tag{B.12}$$



(a) Solution to the Rosenbrock function minimization problem: a few iterations are needed to find the minimizer $\mathbf{x}^\star$.



(b) The contour lines show the local quadratic approximation of R around $\mathbf{x}_0$. The next iterate $\mathbf{x}_1^\star$ is then the minimizer of the quadratic approximation.

Figure B.3: Newton line search. The Wolfe conditions are enforced at each iteration to ensure sufficient decrease of the objective function.

Algorithm 7 BFGS($\mathbf{x}_0$, ϵ)

$\mathbf{H}_0 \leftarrow \mathbf{I}$, $k \leftarrow 0$
while $\|\nabla f_k\| > \epsilon$ **do**
  // Search direction
  $\mathbf{p}_k \leftarrow -\mathbf{H}_k \nabla f_k$
  // Line search with Wolfe conditions
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{s}_k \leftarrow \mathbf{x}_{k+1} - \mathbf{x}_k$
  $\mathbf{y}_k \leftarrow \nabla f_{k+1} - \nabla f_k$
  $\rho_k \leftarrow \dfrac{1}{\mathbf{y}_k^\top \mathbf{s}_k}$
  // BFGS update of the Hessian inverse
  $\mathbf{H}_{k+1} \leftarrow (\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^\top)\mathbf{H}_k(\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^\top) + \rho_k \mathbf{s}_k \mathbf{s}_k^\top$
  $k \leftarrow k + 1$
end while

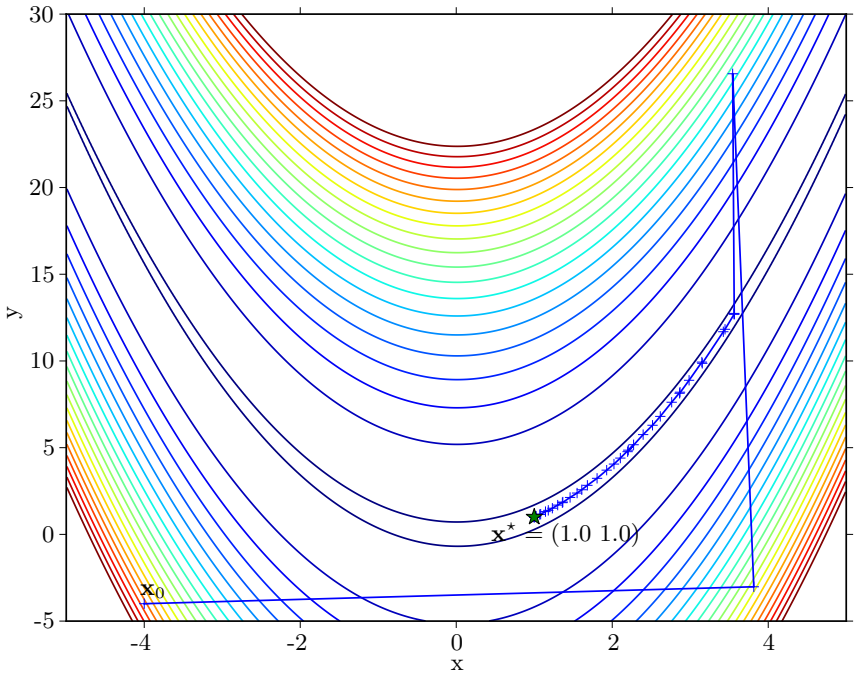


Figure B.4: Rosenbrock function minimization with a quasi-Newton line search using a BFGS-Hessian approximation. It is interesting to note that the line search behaves like a steepest-descent line search for the first iterations, converging towards a Newton line search.

B.1.6 Constrained Optimization

The previous section described methods that allow us to solve the problem of unconstrained optimization, which consists in finding the minimizer $\mathbf{x}^\star \in \mathbb{R}^n$ of a scalar function f. In many cases, however, the minimizer is required to additionally obey to a number of *constraints* in order to be feasible. The general *constrained optimization* problem can then be written as:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \text{ such that } \begin{cases} c_i(\mathbf{x}) = 0, & i \in \mathcal{E} \\ c_i(\mathbf{x}) \geq 0, & i \in \mathcal{I} \end{cases} \tag{B.13}$$

$\Omega = \{\mathbf{x} \in \mathbb{R}^n : c_i(\mathbf{x}) = 0, i \in \mathcal{E}; c_i(\mathbf{x}) \geq 0, i \in \mathcal{I}\}$ is called the *feasible set*, and a point $\mathbf{x}$ is said to be feasible iff $\mathbf{x} \in \Omega$. At a feasible point, an inequality constraint c_i $(i \in \mathcal{I})$ is:

- active iff $c_i(\mathbf{x}) = 0$, i.e. $\mathbf{x}$ is on the boundary of c_i,
- inactive iff $c_i(\mathbf{x}) > 0$, i.e. $\mathbf{x}$ is an interior point of c_i.

We can then define the active set at $\mathbf{x}$:

$$\mathcal{A}(\mathbf{x}) = \mathcal{E} \cup \{i \in \mathcal{I} : c_i(\mathbf{x} = 0)\}, \tag{B.14}$$

as the set of all active constraints at a point $\mathbf{x}$. Let us also define the active constraint gradients at $\mathbf{x}$:

$$\mathbf{A}(\mathbf{x}) = [\nabla c_i(\mathbf{x})]^\top_{i\in\mathcal{A}(\mathbf{x})} \tag{B.15}$$

The optimality conditions we introduced in Theorems 3 and 4 do not hold anymore as we need to take constraints into account. Let us define the *Lagrangian*:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i\in\mathcal{E}\cup\mathcal{I}} \lambda_i c_i(\mathbf{x}), \tag{B.16}$$

where $\lambda_i,\ i \in \mathcal{E} \cup \mathcal{I}$ are called the *Lagrange multipliers*.

It can be shown that candidate minimizers are the stationary points of $\mathcal{L}$, which is equivalent to ∇f lying in the subspace spanned by the active constraint gradients $\nabla c_i,\ i \in \mathcal{A}(\mathbf{x})$. The components of ∇f expressed in this subspace basis correspond then to the active constraint Lagrange multipliers $\lambda_i,\ i \in \mathcal{A}(\mathbf{x})$. This leads to the *Karush-Kuhn-Tucker (KKT) first-order necessary conditions*:

Theorem 5. *$\mathbf{x}^\star$ is a local minimizer, f is C^1 in an open neighborhood of $\mathbf{x}^\star$, $\mathbf{A}(\mathbf{x})$ is full-rank $\Rightarrow \exists!\boldsymbol{\lambda}^\star \in \mathbb{R}^m$ such that:*

$$\begin{array}{lrcll} a) & \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^\star, \boldsymbol{\lambda}^\star) & = & \mathbf{0} & \\ b) & c_i(\mathbf{x}^\star) & = & 0 & \forall i \in \mathcal{E} \\ c) & c_i(\mathbf{x}^\star) & \geq & 0 & \forall i \in \mathcal{I} \\ d) & \lambda_i^\star & \geq & 0 & \forall i \in \mathcal{I} \\ e) & \lambda_i^\star c_i(\mathbf{x}^\star) & = & 0 & \forall i \in \mathcal{E} \cup \mathcal{I} \end{array}$$

Note that $f(\mathbf{x}^\star) = \mathcal{L}(\mathbf{x}^\star, \boldsymbol{\lambda}^\star)$ because of the complementarity condition e) in Equation (5).

B.2 Quadratic Programming

In order to find points which satisfy the KKT conditions, we first consider the simpler problem of *Quadratic programming (QP)*, where the objective function is quadratic, and all constraints are linear:

$$\min_{\mathbf{x}\in\mathbb{R}^n} q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{G}\mathbf{x}+\mathbf{c}^\top\mathbf{x}, \text{ such that } \begin{cases} \mathbf{a}_i^\top\mathbf{x} = b_i, & i \in \mathcal{E} \\ \mathbf{a}_i^\top\mathbf{x} \geq b_i, & i \in \mathcal{I} \end{cases} \quad \mathbf{G} \text{ symmetric.} \tag{B.17}$$

B.2.1 Equality-constrained QP

In the particular case of an equality-constrained QP, we have:

$$\min_{\mathbf{x}\in\mathbb{R}^n} q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{G}\mathbf{x} + \mathbf{c}^\top\mathbf{x}, \text{ such that } \mathbf{A}\mathbf{x} = b \qquad \mathbf{A} \text{ full rank.} \tag{B.18}$$

The KKT conditions given in Theorem (5) imply that the solution $\mathbf{x}^\star$ verifies:

$$\begin{pmatrix} \mathbf{G} & -\mathbf{A}^\top \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^\star \\ \boldsymbol{\lambda}^\star \end{pmatrix} = \begin{pmatrix} -\mathbf{c} \\ \mathbf{b} \end{pmatrix} \overset{\mathbf{x}^\star=\mathbf{x}+\mathbf{p}}{\Longleftrightarrow} \begin{pmatrix} \mathbf{G} & \mathbf{A}^\top \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \begin{pmatrix} -\mathbf{p} \\ \boldsymbol{\lambda}^\star \end{pmatrix} = \begin{pmatrix} \mathbf{g} \\ \mathbf{h} \end{pmatrix} \quad \begin{cases} \mathbf{h} = \mathbf{A}\mathbf{x} - \mathbf{b} \\ \mathbf{g} = \mathbf{c} + \mathbf{G}\mathbf{x} \\ \mathbf{p} = \mathbf{x}^\star - \mathbf{x} \end{cases} \tag{B.19}$$

The final linear system in Equation (B.19) is called the *KKT system*. It can be solved using standard linear algebra techniques, and its solution gives the update $\mathbf{p}$ which leads directly to the minimizer $\mathbf{x}^\star$ of q.

We use this method to solve an equality-constrained QP, shown in Figure B.5: for the same quadratic objective function, we find the minimizer for three different linear equality constraints. We can see that the associated Lagrange multiplier is higher as the constrained minimizer is further from the unconstrained minimum. Intuitively, Lagrange multiplier give an idea of the "force" which the constraints are exerting on the constrained minimizer to keep it away from the unconstrained minimum.

B.2.2 Inequality-Constrained QP

Now that we can solve an equality-constrained QP, we can move on to solve the general QP presented in Equation (B.17). Several types of method such as active-set, gradient-projection and interior-point can be used to this end. We describe here the active-set method for solving inequality-constrained QP.

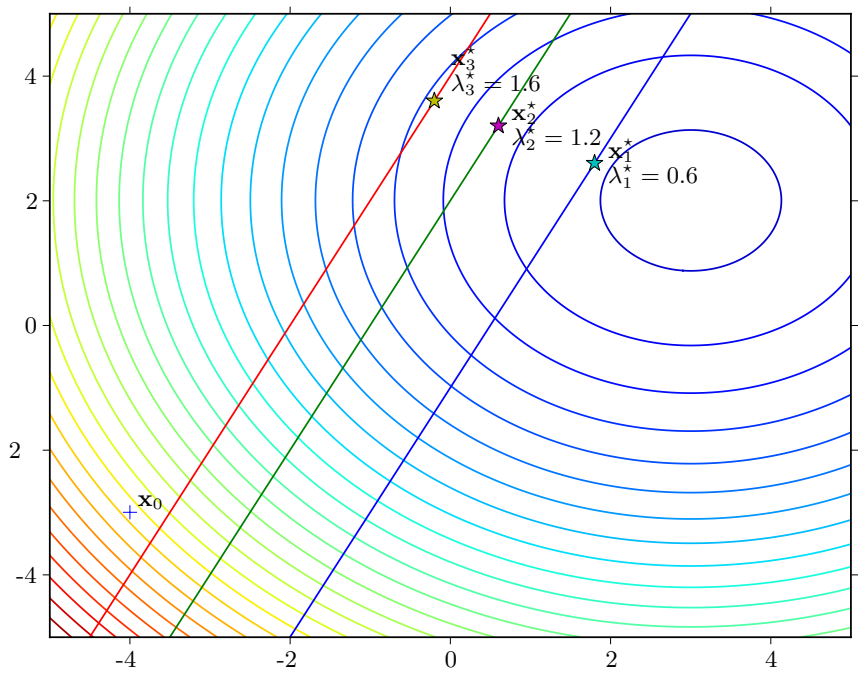


Figure B.5: Solution of a constrained QP problem. One linear equality constraint is used, and solutions for different values of the constraint are shown. Note that the farther the constraint is from the unconstrained minimum, the larger is the associated Lagrange multiplier.

The active-set method, described in detail in Algorithm 8, consists in iteratively estimating the optimal active set, solving the underlying equality-constrained QP with the active constraints, and repeat until the optimal active set is correctly found. We implement an active-set method to solve a QP as shown in Figure B.6. One linear equality constraint and two linear inequality constraints are added. Starting from a point $\mathbf{x}_0$, the optimal solution and active set are found iteratively.

B.3 Nonlinear Programming

We are now ready to tackle *Nonlinear programming (NLP)*, i.e. to solve the general problem from Equation (B.13). Two of the most successful methods used nowadays to solve large-scale problems are *Sequential Quadratic Programming (SQP)* and *Interior-Point Methods (IPM)*.

B.3.1 Sequential Quadratic Programming

The idea behind SQP is quite simple: for each SQP iteration, we build a local quadratic approximation of the objective function and linearized approximation of the constraints in order to obtain a *QP subproblem*:

Algorithm 8 `ActiveSetSolve`($\mathbf{x}_0$)

// Initialize active set
$\mathcal{W}_0 \leftarrow$ subset of the active constraints at $\mathbf{x}_0$
for $k = 0, 1, 2, \ldots$ **do**
  // Find minimizer of equality-constrained QP using active constraints
  $(\mathbf{p_k}\ \boldsymbol{\lambda}_{k+1}) \leftarrow$ `SolveKKTSystem`($\mathbf{W_k}$)
  if $\mathbf{p}_k = \mathbf{0}$ **then**
    if $\lambda_{k+1,i} \geq 0\ \forall i \in \mathcal{W}_k \cap \mathcal{I}$ **then**
      // All inequality constraints in the active set are active
      return $(\mathbf{x}_k, \boldsymbol{\lambda}_{k+1})$
    else
      // At least one inequality constraint in the active set is inactive
      // Remove the one that is "least active"
      $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \backslash \{\arg\min_{j \in \mathcal{W}_k \cap \mathcal{I}} \lambda_{k+1,j}\}$
    end if
  else
    // Compute α_k such that the constraints which are *not* in the active set are not violated
    $\alpha_k \leftarrow \min_{\substack{i \notin \mathcal{W}_k \\ \mathbf{a}_i^\top \mathbf{p}_k < 0}} \dfrac{b_i - \mathbf{a}_i^\top \mathbf{x}_k}{\mathbf{a}_i^\top \mathbf{p}_k}$
    $\mathbf{x}_{k+1} \leftarrow \alpha_k \mathbf{p}_k$
    if $\alpha_k < 1$ **then**
      // At least one constraint which is not the active set is active, add one
      $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \cup \{\text{one blocking constraint}\}$
    else
      // Keep the same active set
      $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$
    end if
  end if
end for

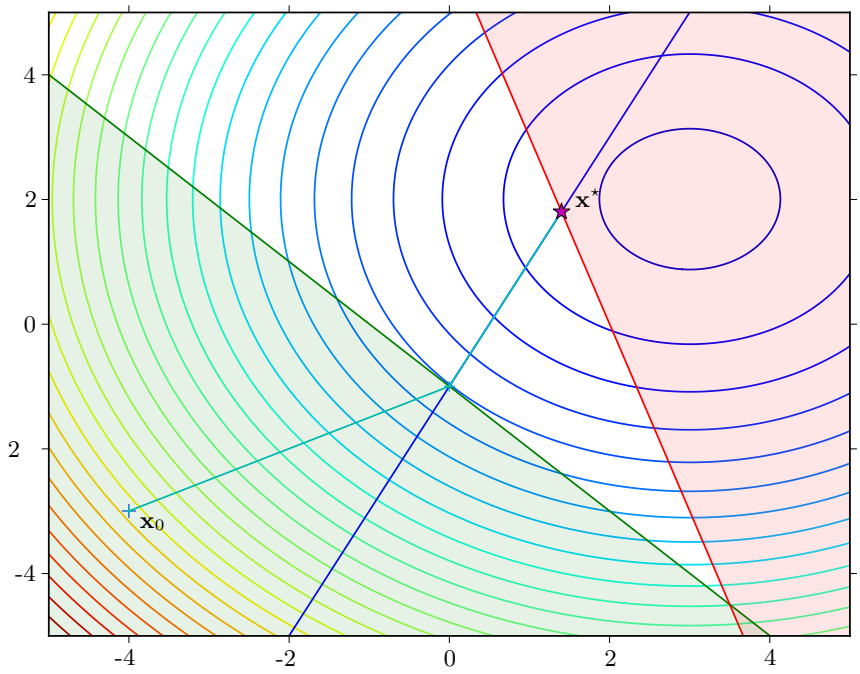


Figure B.6: Solution to the general QP problem using an active-set method. The blue line represents the linear equality constraint, while the green and red half-planes represent the linear inequalities (the infeasible set is filled). Starting from $\mathbf{x}_0$, the optimal solution $\mathbf{x}^\star$ and active set are found iteratively.

$$\min_{\mathbf{p}} \frac{1}{2}\mathbf{p}_k^\top \nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)\mathbf{p}_k + \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k + f(\mathbf{x}_k)$$
$$\text{s.t.} \begin{cases} \nabla c_i(\mathbf{x}_k)^\top \mathbf{p}_k + c_i(\mathbf{x}_k) = 0, & i \in \mathcal{E} \\ \nabla c_i(\mathbf{x}_k)^\top \mathbf{p}_k + c_i(\mathbf{x}_k) \geq 0, & i \in \mathcal{I} \end{cases} \tag{B.20}$$

We then solve the QP subproblem, with active-set methods for instance, and repeat iteratively until the convergence test is satisfied, as described in Algorithm 9.

Algorithm 9 `SQPSolve`($\mathbf{x}_0$,$\boldsymbol{\lambda}_0$, ϵ)

for $k = 0, 1, 2, ...$ **do**
  Evaluate f_k, ∇f_k, $c_i(\mathbf{x}_k)$, $\nabla c_i(\mathbf{x}_k)$, $\nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)$
  $(\mathbf{p}_k, \boldsymbol{\lambda}_{k+1}) \leftarrow$ `ActiveSetSolve`$(\mathbf{x}_k)$
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  if convergence test satisfied for tolerance ϵ **then**
    return $(\mathbf{x}_{k+1}, \boldsymbol{\lambda}_{k+1})$
  end if
end for

This requires us to derive the Hessian of both objective function and constraints, see Equation (B.20). One solution to avoid computing them exactly is to use the BFGS method as seen in Section B.1.5. Note that iterates can be infeasible and that only the final solution is guaranteed to be feasible. Also, as in the case of line search methods in unconstrained optimization, there is no guarantee that the solution of QP subproblem will lead to a decrease of the original objective function, or that it will improve feasibility with respect to the constraints. A *merit function* can be used to this end and ensure, just like the Wolfe conditions for the unconstrained case, that there is a sufficient decrease of f and forbid too great infeasibility. One simple merit function is the ℓ_1 exact function defined as:

$$\phi_1(\mathbf{x}, \mu) = f(\mathbf{x}) + \mu \sum_{i \in \mathcal{E}} |c_i(\mathbf{x})| + \mu \sum_{i \in \mathcal{I}} [c_i(\mathbf{x})]^-, \quad \mu > 0, \quad [x]^- = \max(0, -x). \tag{B.21}$$

In Figure B.7, we show an implementation of SQP using the ℓ_1 merit function, where the Rosenbrock function is minimized under one nonlinear equality constraint and one nonlinear inequality constraint, represented by a circle and a disk respectively. Interestingly, it shows that not all initial values will lead to the global minimizer, as some iterates might get stuck in local minimizers. Also it is clear that not all iterates are feasible, which means that we have to wait until the end of the optimization process (up to a specified tolerance) to retrieve a feasible solution.

B.3.2 Interior-Point Methods for Nonlinear Programming

We describe *interior-point methods*, also known as *log-barrier methods*. They propose an alternative to SQP and are also very powerful when used to solve large-scale NLP.

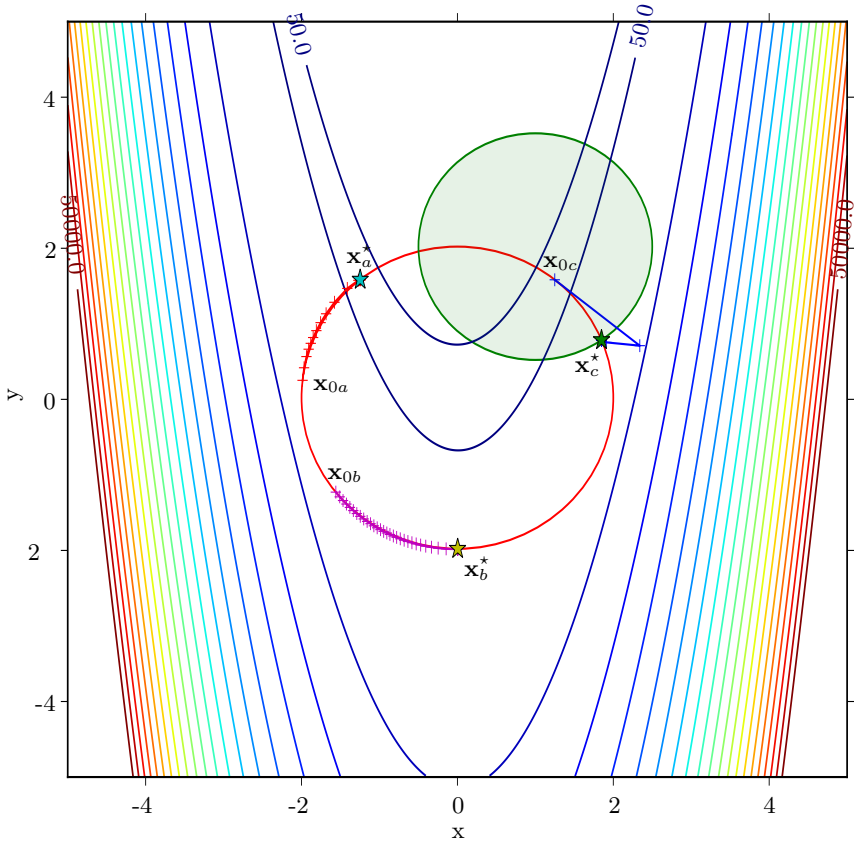


Figure B.7: Minimization of the Rosenbrock function under the nonlinear circle equality constraint $x^2 + y^2 - 4 = 0$ and the disk inequality constraint $(x-1)^2 + (y-2)^2 - 2.25 \geq 0$, starting from three different initial points. Only $\mathbf{x_{0a}}$ leads to the global minimizer $\mathbf{x}_a^\star$, while $\mathbf{x_{0b}}$ and $\mathbf{x_{0c}}$ do not.

Their principle is the following: a slack variable $\mathbf{s}$ is introduced to reformulate the general NLP as:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \text{ such that } \begin{cases} c_i(\mathbf{x}) = 0, & i \in \mathcal{E} \\ c_i(\mathbf{x}) - s_i = 0, & i \in \mathcal{I} \\ s_i \geq 0, & i \in \mathcal{I} \end{cases} \tag{B.22}$$

A parameter $\mu > 0$ is also introduced, which allows us to obtain the *perturbed KKT conditions*:

$$\begin{array}{lrcl} a) & \nabla f(\mathbf{x}) - \mathbf{A}_{\mathcal{E}}^{\top}(\mathbf{x})\mathbf{y} - \mathbf{A}_{\mathcal{I}}^{\top}(\mathbf{x})\mathbf{z} & = & \mathbf{0} \\ b) & \mathbf{c}_{\mathcal{E}}(\mathbf{x}) & = & \mathbf{0} \\ b') & \mathbf{c}_{\mathcal{I}}(\mathbf{x}) - \mathbf{s} & = & \mathbf{0} \\ c) & \mathbf{s} & \geq & \mathbf{0} \\ d) & \mathbf{z} & \geq & \mathbf{0} \\ e) & \mathbf{Sz} & = & \mu\mathbf{e}, \end{array} \tag{B.23}$$

with $\mathbf{c}_{\mathcal{E}}$, $\mathbf{c}_{\mathcal{I}}$ denoting the constraints vector, $\mathbf{A}_{\mathcal{E}}(\mathbf{x})$, $\mathbf{A}_{\mathcal{I}}(\mathbf{x})$ denoting their respective Jacobians, $\mathbf{y}$, $\mathbf{z}$ denoting the Lagrange multipliers, $\mathbf{S} = \text{diag}(s_i)$ and $\mathbf{e}^{\top} = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}$.

The variables $\mathbf{s}$ and $\mathbf{z}$ are eliminated from the KKT conditions:

$$\forall i \in \mathcal{I} \quad \left.\begin{array}{c} s_i z_i - \mu = 0 \\ c_i(\mathbf{x}) - s_i = 0 \end{array}\right\} \Rightarrow z_i = \frac{\mu}{c_i(\mathbf{x})}, \tag{B.24}$$

and substituted in $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in Equation (B.23)a):

$$\begin{array}{c} \nabla f(\mathbf{x}) - \mathbf{A}_{\mathcal{E}}^{\top}(\mathbf{x}) - \sum_{i\in\mathcal{I}} \frac{\mu}{c_i(\mathbf{x})} \nabla c_i(\mathbf{x}) = \mathbf{0} \\ \Updownarrow \\ \min_{\mathbf{x}} P(\mathbf{x}; \mu) = f(\mathbf{x}) - \mu \sum_{i\in\mathcal{I}} \log c_i(\mathbf{x}) \quad \text{such that } \mathbf{c}_{\mathcal{E}} = \mathbf{0} \end{array} \tag{B.25}$$

Finding the stationary points of the Lagrangian of the original problem is then equivalent to minimizing the equality-constrained *log-barrier function* $P(\mathbf{x}; \mu)$ and making $\mu \to 0$.

One way to solve the NLP is to start with an initial value of the parameter μ_0. The associated log-barrier function problem is then solved, and the process is repeated while decreasing μ_k towards 0. The log-barrier function problem solutions for the *central path* $\{\mathbf{x}^{\star}(\mu_k)\}_k$, with the particularity that each iterate is feasible with respect to the original equality and inequality constraints.

B.3.3 Conclusion

To conclude, SQP and IPM are equally efficient for solving large-scale NLP problems. They are somewhat similar as both of them are based on two nested iteration loops. Their main difference is the fact that IPM generate feasible iterates thanks to their

conservative approach, while SQP methods may generate infeasible iterates, with the guarantee that only the solution is feasible.

Bibliography

[Arisumi 08] Hitoshi Arisumi, Sylvain Miossec, J-R Chardonnet & Kazuhito Yokoi. *Dynamic lifting by whole body motion of humanoid robots.* In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 668–675. IEEE, 2008.

[Baerlocher 98] P. Baerlocher & R. Boulic. *Task-priority formulations for the kinematic control of highly redundant articulated structures.* In 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1998. Proceedings., volume 1, 1998.

[Baudouin 11] Léo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse & Eiichi Yoshida. *Real-time replanning using 3D environment for humanoid robot.* In Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on, pages 584–589. IEEE, 2011.

[Bellman 65] Richard Bellman & Robert E Kalaba. Dynamic programming and modern control theory. Academic Press New York, 1965.

[Berenson 11] D. Berenson, S.S. Srinivasa & J. Kuffner. *Task Space Regions: A framework for pose-constrained manipulation planning.* The International Journal of Robotics Research, 2011.

[Betts 98] John T Betts. *Survey of numerical methods for trajectory optimization.* Journal of guidance, control, and dynamics, vol. 21, no. 2, 1998.

[Betts 10] John T Betts. Practical methods for optimal control and estimation using nonlinear programming, volume 19. Society for Industrial and Applied Mathematics, 2010.

[Biegler 09] LT Biegler & VM Zavala. *Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization.* Computers & Chemical Engineering, vol. 33, no. 3, pages 575–582, 2009.

[Bobrow 85] James E Bobrow, Steven Dubowsky & JS Gibson. *Time-optimal control of robotic manipulators along specified paths.* The International Journal of Robotics Research, vol. 4, no. 3, pages 3–17, 1985.

[Bock 84] H.G. Bock & K.J. Plitt. *A Multiple Shooting algorithm for direct solution of optimal control problems.* In Proceedings of the 9th IFAC World Congress, pages 242–247, Budapest, 1984. Pergamon Press. Available at http://www.iwr.uni-heidelberg.de/groups/agbock/FILES/Bock1984.pdf.

[Boltyanskii 60] Vladimir Grigor'evich Boltyanskii, Revaz Valer'yanovich Gamkrelidze & Lev Semenovich Pontryagin. *The theory of optimal processes. I. The maximum principle.* Rapport technique, DTIC Document, 1960.

[Bonnans 06] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal & C.A. Sagastizábal. Numerical optimization: theoretical and practical aspects. Springer, 2006.

[Bouyarmane 12] K. Bouyarmane & A. Kheddar. *Humanoid Robot Locomotion and Manipulation Step Planning.* Advanced Robotics, vol. 26, no. 10, pages 1099–1126, 2012.

[Bretl 06] T. Bretl. *Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem.* The International Journal of Robotics Research, vol. 25, no. 4, pages 317–342, 2006.

[Chestnutt 05] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins & T. Kanade. *Footstep Planning for the Honda ASIMO Humanoid.* In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pages 629 – 634, April 2005.

[Chevallereau 01] C Chevallereau & Y Aoustin. *Optimal reference trajectories for walking and running of a biped robot.* Robotica, vol. 19, no. 5, pages 557–569, 2001.

[Choset 05] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki & Sebastian Thrun. Principles of robot motion: Theory, algorithms, and implementations. MIT Press, Cambridge, MA, June 2005.

[Coros 10] Stelian Coros, Philippe Beaudoin & Michiel van de Panne. *Generalized biped walking control.* ACM Transactions on Graphics (TOG), vol. 29, no. 4, page 130, 2010.

[Dalibard 09] S. Dalibard, A. Nakhaei, F. Lamiraux & J.-P. Laumond. *Whole-body task planning for a humanoid robot: a way to integrate collision avoidance.* In Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on, pages 355 –360, 7-10 2009.

[Dang 12] N. Dang, F. Lamiraux & J.-P. Laumond. *Experiments on whole-body manipulation and locomotion with footstep real-time optimization.* In IEEE International Conference on Humanoid Robots (Humanoids), Osaka, 2012.

[Dellin 12] Christopher M Dellin & Siddhartha S Srinivasa. *A framework for extreme locomotion planning.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 989–996. IEEE, 2012.

[Diankov 08] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa & J. Kuffner. *Bispace planning: Concurrent multi-space exploration.* Proceedings of Robotics: Science and Systems IV, 2008.

[Diehl 06] Moritz Diehl, Hans Georg Bock, Holger Diedam & P-B Wieber. *Fast direct multiple shooting algorithms for optimal robot control.* Fast Motions in Biomechanics and Robotics, pages 65–93, 2006.

[Dubins 57] L. E. Dubins. *On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents.* American Journal of Mathematics, vol. 79, no. 3, pages pp. 497–516, 1957.

[Dubowsky 86] S Dubowsky, M Norris & Z Shiller. *Time optimal trajectory planning for robotic manipulators with obstacle avoidance: a CAD approach.* In Robotics and Automation. Proceedings. 1986 IEEE International Conference on, volume 3, pages 1906–1912. IEEE, 1986.

[Eberly 07] David H Eberly. 3d game engine design: a practical approach to real-time computer graphics. Morgan Kaufmann Publishers, 2007.

[Eberly 11] David H. Eberly. *Wild Magic 5.7.* `http://www.geometrictools.com`, 2011.

[Ericson 04] Christer Ericson. Real-time collision detection. Morgan Kaufmann, 2004.

[Escande 07] Adrien Escande, Sylvain Miossec & Abderrahmane Kheddar. *Continuous gradient proximity distance for humanoids free-collision optimized-postures.* In Humanoid Robots, 2007 7th

IEEE-RAS International Conference on, pages 188–195. IEEE, 2007.

[Escande 13] Adrien Escande, Abderrahmane Kheddar & Sylvain Miossec. *Planning contact points for humanoid robots.* Robotics and Autonomous Systems, 2013.

[Featherstone 08] R. Featherstone. Rigid body dynamics algorithms, volume 49. Springer Berlin:, 2008.

[Felis 12] Martin L Felis, Katja Mombaur, Hideki Kadone & Alain Berthoz. *Modeling and identification of emotional aspects of locomotion.* Journal of Computational Science, 2012.

[Felzenszwalb 04] Pedro Felzenszwalb & Daniel Huttenlocher. *Distance transforms of sampled functions.* Technical Report TR2004-1963, 2004.

[Ferre 04] E. Ferre & J.P. Laumond. *An iterative diffusion algorithm for part disassembly.* In 2004 International Conference on Robotics and Automation (ICRA'2004), pages 3149–3154, New Orleans (USA), 2004.

[Fitzpatrick 06] P. Fitzpatrick. Advanced calculus, volume 5. American Mathematical Society, 2006.

[Flash 85] Tamar Flash & Neville Hogans. *The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model.* Journal of neuroscience, 1985.

[Garimort 11] Johannes Garimort & Armin Hornung. *Humanoid navigation with dynamic footstep plans.* In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 3982–3987. IEEE, 2011.

[Geraerts 07] Roland Geraerts & Mark H Overmars. *Creating high-quality paths for motion planning.* The International Journal of Robotics Research, vol. 26, no. 8, pages 845–863, 2007.

[Gill 02] Philip E Gill, Walter Murray & Michael A Saunders. *SNOPT: An SQP algorithm for large-scale constrained optimization.* SIAM journal on optimization, vol. 12, no. 4, pages 979–1006, 2002.

[Glassman 10] Elena Glassman & Russ Tedrake. *A quadratic regulator-based heuristic for rapidly exploring state space.* In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 5021–5028. IEEE, 2010.

[Goodman 04] J.E. Goodman & J. O'Rourke. Handbook of discrete and computational geometry. Chapman & Hall/CRC, 2004.

[Gottschalk 96] S. Gottschalk, M.C. Lin & D. Manocha. *OBBTree: a hierarchical structure for rapid interference detection.* In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 171–180. ACM, 1996.

[Guizzo 10] E. Guizzo. *France Developing Advanced Humanoid Robot Romeo.* IEEE Spectrum Automaton Blog, December 13, 2010.

[Harada 03] Kensuke Harada, Shuuji Kajita, Kenji Kaneko & Hirohisa Hirukawa. *Zmp analysis for arm/leg coordination.* In Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, volume 1, pages 75–81. IEEE, 2003.

[Hart 68] Peter Hart, Nils Nilsson & Bertram Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pages 100–107, February 1968.

[Hauser 10a] K. Hauser & J.C. Latombe. *Multi-modal motion planning in non-expansive spaces.* The International Journal of Robotics Research, vol. 29, no. 7, pages 897–915, 2010.

[Hauser 10b] Kris Hauser & Victor Ng-Thow-Hing. *Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts.* In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 2493–2498. IEEE, 2010.

[Herdt 10] Andrei Herdt, Holger Diedam, Pierre-Brice Wieber, Dimitar Dimitrov, Katja Mombaur & Moritz Diehl. *Online Walking Motion Generation with Automatic Footstep Placement.* Advanced Robotics, vol. 24, no. 5-6, pages 719–737, 2010.

[Hicks 71] GA Hicks & WH Ray. *Approximation methods for optimal control synthesis.* The Canadian Journal of Chemical Engineering, vol. 49, no. 4, pages 522–528, 1971.

[Hirukawa 06] Hirohisa Hirukawa, Shizuko Hattori, Kensuke Harada, Shuuji Kajita, Kenji Kaneko, Fumio Kanehiro, Kiyoshi Fujiwara & Mitsuharu Morisawa. *A universal stability criterion of the foot contact of legged robots-adios ZMP.* In Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pages 1976–1983. IEEE, 2006.

[Hornung 10] Armin Hornung, Kai M Wurm & Maren Bennewitz. *Humanoid robot localization in complex indoor environments.* In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 1690–1695. IEEE, 2010.

[Houska 10] Boris Houska, Hans Joachim Ferreau & Moritz Diehl. *ACADO toolkitAn open-source framework for automatic control and dynamic optimization.* Optimal Control Applications and Methods, vol. 32, no. 3, pages 298–312, 2010.

[Hudson 97] T.C. Hudson, M.C. Lin, J. Cohen, S. Gottschalk & D. Manocha. *V-COLLIDE: accelerated collision detection for VRML.* In Proceedings of the second symposium on Virtual reality modeling language, pages 117–ff. ACM, 1997.

[Kajita 03] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi & H. Hirukawa. *Biped walking pattern generation by using preview control of zero-moment point.* In Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on, volume 2, Sept. 2003.

[Kalakrishnan 11] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor & Stefan Schaal. *STOMP: Stochastic trajectory optimization for motion planning.* In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 4569–4574. IEEE, 2011.

[Kanehiro 08] Fumio Kanehiro, Florent Lamiraux, Oussama Kanoun, Eiichi Yoshida & Jean-Paul Laumond. *A local collision avoidance method for non-strictly convex polyhedra.* Proceedings of robotics: science and systems IV, 2008.

[Kaneko 04] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi & T. Isozumi. *Humanoid robot HRP-2.* In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, volume 2, pages 1083–1090. IEEE, 2004.

[Kanoun 09] Oussama Kanoun, Florent Lamiraux, Pierre-Brice Wieber, Fumio Kanehiro, Eiichi Yoshida & Jean-Paul Laumond. *Prioritizing linear equality and inequality systems: Application to local motion planning for redundant robots.* In Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, pages 2939 –2944, May 2009.

[Kanoun 11] Oussama Kanoun. *Real-time prioritized kinematic control under inequality constraints for redundant manipulators.* In Proceedings of Robotics: Science and Systems, Los Angeles, CA, USA, 2011.

[Karaman 11] Sertac Karaman & Emilio Frazzoli. *Sampling-based algorithms for optimal motion planning.* The International Journal of Robotics Research, 2011.

[Kavraki 96] L.E. Kavraki, P. Svestka, J.-C. Latombe & M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* Robotics and Automation, IEEE Transactions on, vol. 12, no. 4, pages 566 –580, August 1996.

[Khatib 85] O. Khatib. *Real-time obstacle avoidance for manipulators and mobile robots.* In Robotics and Automation. Proceedings. 1985 IEEE International Conference on, volume 2, pages 500 – 505, March 1985.

[Khatib 04] O. Khatib, L. Sentis, J. Park & J. Warren. *Whole body dynamic behavior and control of human-like robots.* International Journal of Humanoid Robotics, vol. 1, no. 1, pages 29–43, 2004.

[Kim 02] Young J. Kim, Miguel A. Otaduy, Ming C. Lin & Dinesh Manocha. *Fast penetration depth computation for physically-based animation.* In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '02, pages 23–31, New York, NY, USA, 2002. ACM.

[Koch 12] Kai Henning Koch, Katja Mombaur & Philippe Soueres. *Optimization-based walking generation for humanoid robot.* In Robot Control, volume 10, pages 498–504, 2012.

[Kuffner 00] Jr. Kuffner J.J. & S.M. LaValle. *RRT-connect: An efficient approach to single-query path planning.* In Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on, volume 2, pages 995 –1001 vol.2, 2000.

[Kuffner 01] Jr. Kuffner J.J., K. Nishiwaki, S. Kagami, M. Inaba & H. Inoue. *Footstep planning among obstacles for biped robots.* In Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, volume 1, pages 500 –505 vol.1, 2001.

[Larsen 00] Eric Larsen, Stefan Gottschalk, Ming C Lin & Dinesh Manocha. *Fast distance queries with rectangular swept sphere volumes.* In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, volume 4, pages 3719–3726. IEEE, 2000.

[Latombe 91] Jean-Claude Latombe. Robot motion planning. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[Laumond 94] J.-P. Laumond, P.E. Jacobs, M. Taix & R.M. Murray. *A motion planner for nonholonomic mobile robots.* Robotics and Automation, IEEE Transactions on, vol. 10, no. 5, pages 577 –593, oct 1994.

[Laumond 98] J.-P. Laumond. Robot motion planning and control. Springer, 1998.

[Laumond 06] J.-P. Laumond. *Kineo CAM: a success story of motion planning algorithms.* Robotics & Automation Magazine, IEEE, vol. 13, no. 2, pages 90–93, 2006.

[Lauterbach 10] Christian Lauterbach, Qi Mo & Dinesh Manocha. *gProximity: Hierarchical GPU-based Operations for Collision and Distance Queries.* In Computer Graphics Forum, volume 29, pages 419–428. Wiley Online Library, 2010.

[LaValle 06] S. M. LaValle. Planning algorithms. Cambridge University Press, Cambridge, U.K., 2006.

[Lee 12] Youngeun Lee, Sébastien Lengagne, Abderrahmane Kheddar & Young J. Kim. *Accurate evaluation of a distance function for optimization-based motion planning.* In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 1513–1518. IEEE, 2012.

[Leineweber 03a] Daniel B Leineweber, Irene Bauer, Hans Georg Bock & Johannes P Schlöder. *An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part 1: theoretical aspects.* Computers & Chemical Engineering, vol. 27, no. 2, pages 157–166, 2003.

[Leineweber 03b] Daniel B Leineweber, Andreas Schäfer, Hans Georg Bock & Johannes P Schlöder. *An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization: Part II: Software aspects and applications.* Computers & chemical engineering, vol. 27, no. 2, pages 167–174, 2003.

[Lengagne 13] Sébastien Lengagne, Joris Vaillant, Eiichi Yoshida & Abderrahmane Kheddar. *Generation of Whole-body Optimal Dynamic Multi-Contact Motions.* The International Journal of Robotics Research, 2013.

[Lozano-Perez 83] T. Lozano-Perez. *Spatial Planning: A Configuration Space Approach*. Computers, IEEE Transactions on, vol. C-32, no. 2, pages 108 –120, feb. 1983.

[Mainprice 12] Jim Mainprice. *Planification de mouvement pour la manipulation d'objets sous contraintes d'interaction homme-robot*. PhD thesis, INSA de Toulouse, 2012.

[Miossec 06] Sylvain Miossec, Kazuhito Yokoi & Abderrahmane Kheddar. *Development of a software for motion optimization of robots-application to the kick motion of the hrp-2 robot*. In Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on, pages 299–304. IEEE, 2006.

[Mombaur 05] Katja D Mombaur, Hans Georg Bock, Johannes P Schlöder & Richard W Longman. *Open-loop stable solutions of periodic optimal control problems in robotics*. ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, vol. 85, no. 7, pages 499–515, 2005.

[Mombaur 10] Katja Mombaur, Anh Truong & Jean-Paul Laumond. *From human to humanoid locomotion–an inverse optimal control approach*. Auton. Robots, vol. 28, pages 369–383, April 2010.

[Mordatch 12] Igor Mordatch, Emanuel Todorov & Zoran Popović. *Discovery of complex behaviors through contact-invariant optimization*. ACM Transactions on Graphics (TOG), vol. 31, no. 4, page 43, 2012.

[Moulard 09] Thomas Moulard. *RobOptim*. `https://github.com/laas/roboptim`, 2009.

[Moulard 10] Thomas Moulard, Florent Lamiraux & Pierre-Brice Wieber. Collision-free walk planning for humanoid robots using numerical optimization. Retrieved from http://hal.archives-ouvertes.fr/hal-00486997/en/, 2010.

[Moulard 12a] Thomas Moulard. *Optimisation numérique pour la robotique et exécution de trajectoires référencées capteurs*. PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2012.

[Moulard 12b] Thomas Moulard, Florent Lamiraux & Olivier Stasse. *Trajectory following for legged robots*. In Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on, pages 657–662. IEEE, 2012.

[Nakamura 86] Y. Nakamura & H. Hanafusa. *Inverse kinematic solutions with singularity robustness for robot manipulator control*. ASME,

Transactions, Journal of Dynamic Systems, Measurement, and Control, vol. 108, pages 163–171, 1986.

[Nakhaei 08] A. Nakhaei & F. Lamiraux. *Motion planning for humanoid robots in environments modeled by vision.* In Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on, pages 197 –204, dec. 2008.

[Nocedal 99] Jorge Nocedal & Stephen J Wright. Numerical optimization. Springer verlag, 1999.

[Pan 12] Jia Pan, Sachin Chitta & Dinesh Manocha. *FCL: A general purpose library for collision and proximity queries.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 3859–3866. IEEE, 2012.

[Perez 12] Alejandro Perez, R Platt, George Konidaris, Leslie Kaelbling & Tomas Lozano-Perez. *LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 2537–2542. IEEE, 2012.

[Perrin 11] N. Perrin, O. Stasse, F. Lamiraux & E. Yoshida. *Weakly collision-free paths for continuous humanoid footstep planning.* In IEEE International Conference on Intelligent Robots and Systems (IROS'11), 2011.

[Perrin 12a] N. Perrin. *From Discrete to Continuous Motion Planning.* In Tenth International Workshop on the Algorithmic Foundations of Robotics (WAFR'12), 2012.

[Perrin 12b] N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux & E. Yoshida. *Fast humanoid robot collision-free footstep planning using swept volume approximations.* IEEE Transactions on Robotics (T-RO), vol. 28, no. 2, 2012.

[Porta 12] J.M. Porta, L. Jaillet & O. Bohigas. *Randomized path planning on manifolds based on higher-dimensional continuation.* The International Journal of Robotics Research, vol. 31, no. 2, pages 201–215, 2012.

[Posa 12] Michael Posa & Russ Tedrake. *Direct trajectory optimization of rigid body dynamical systems through contact.* In Workshop on the Algorithmic Foundations of Robotics, 2012.

[Ratliff 09] N. Ratliff, M. Zucker, J.A. Bagnell & S. Srinivasa. *Chomp: Gradient optimization techniques for efficient motion planning.* In

Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 489–494. IEEE, 2009.

[Rosenbrock 60] Howard H Rosenbrock. *An automatic method for finding the greatest or least value of a function.* The Computer Journal, vol. 3, no. 3, pages 175–184, 1960.

[Saab 12] L. Saab, O. Ramos, N. Mansard, P. Souères & J-Y. Fourquet. *Dynamic Whole-Body Motion Generation Under Rigid Contacts and Other Unilateral Constraints.* IEEE Transaction on Robotics, November 2012. (in press).

[Sargent 78] R Sargent & G Sullivan. *The development of an efficient optimal control package.* Optimization Techniques, pages 158–168, 1978.

[Schneider 03] Philip J Schneider & David H Eberly. Geometric tools for computer graphics. Morgan Kaufmann Pub, 2003.

[Schultz 10] G. Schultz & K. Mombaur. *Modeling and Optimal Control of Human-Like Running.* Mechatronics, IEEE/ASME Transactions on, 2010.

[Shkolnik 11] A. Shkolnik, M. Levashov, I.R. Manchester & R. Tedrake. *Bounding on rough terrain with the LittleDog robot.* The International Journal of Robotics Research, vol. 30, no. 2, page 192, 2011.

[Siciliano 91] B. Siciliano & J.J.E. Slotine. *A general framework for managing multiple tasks in highly redundant robotic systems.* In Advanced Robotics, 1991.'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on, pages 1211–1216, 1991.

[Sirisena 81] H. R. Sirisena & F. S. Chou. *State parameterization approach to the solution of optimal control prob lems.* Optimal Control Applications and Methods, vol. 2, no. 3, pages 289–298, 1981.

[Stasse 08] Olivier Stasse, Adrien Escande, Nicolas Mansard, Sylvain Miossec, Paul Evrard & Abderrahmane Kheddar. *Real-time (self)-collision avoidance task on a hrp-2 humanoid robot.* In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 3200–3205. IEEE, 2008.

[Sucan 11] Ioan Sucan. *planning_environment.* `http://www.ros.org/wiki/planning_environment`, 2011.

[Suleiman 08] Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, J-P Laumond & André Monin. *On human motion imitation by humanoid robot.* In

Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 2697–2704. IEEE, 2008.

[Suleiman 10] W. Suleiman, F. Kanehiro, E. Yoshida, J.-P. Laumond & A. Monin. *Time Parameterization of Humanoid-Robot Paths.* Robotics, IEEE Transactions on, vol. 26, no. 3, pages 458 –468, june 2010.

[Tassa 12] Yuval Tassa, Tom Erez & Emanuel Todorov. *Synthesis and stabilization of complex behaviors through online trajectory optimization.* In Intelligent Robots and Systems, 2012. Proceedings. 2012 IEEE/RSJ International Conference on. IEEE/RSJ, 2012.

[Todorov 06] Emanuel Todorov. *Optimal control theory.* Bayesian brain: probabilistic approaches to neural coding, pages 269–298, 2006.

[Toussaint 07] M. Toussaint, M. Gienger & C. Goerick. *Optimization of sequential attractor-based movement for compact behaviour generation.* In Humanoid Robots, 2007 7th IEEE-RAS International Conference on, pages 122 –129, 29 2007-dec. 1 2007.

[Trinkle 97] Jeffrey C Trinkle, J-S Pang, Sandra Sudarsky & Grace Lo. *On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction.* ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, vol. 77, no. 4, pages 267–279, 1997.

[Tsang 75] TH Tsang, DM Himmelblau & TF Edgar. *Optimal control via collocation and non-linear programming.* International Journal of Control, vol. 21, no. 5, pages 763–768, 1975.

[Verscheure 09] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter & Moritz Diehl. *Time-optimal path tracking for robots: A convex optimization approach.* Automatic Control, IEEE Transactions on, vol. 54, no. 10, pages 2318–2327, 2009.

[Vukobratovic 69] M. Vukobratovic & D. Juricic. *Contribution to the synthesis of biped gait.* Biomedical Engineering, IEEE Transactions on, no. 1, pages 1–6, 1969.

[Wieber 02] Pierre-Brice Wieber. *On the stability of walking systems.* In Proceedings of the International Workshop on Humanoid and Human Friendly Robotics, Tsukuba, Japan, 2002.

[Xia 09] Zeyang Xia, Guodong Chen, Jing Xiong, Qunfei Zhao & Ken Chen. *A random sampling-based approach to goal-directed footstep*

planning for humanoid robots. In Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on, July 2009.

[Yoshida 06] E. Yoshida, O. Kanoun, C. Esteves & J.P. Laumond. *Task-driven support polygon reshaping for humanoids.* In Humanoid Robots, 2006 6th IEEE-RAS International Conference on, pages 208–213, 2006.

[Yoshida 08] E. Yoshida, C. Esteves, I. Belousov, J.-P. Laumond, T. Sakaguchi & K. Yokoi. *Planning 3-D Collision-Free Dynamic Robotic Motion Through Iterative Reshaping.* Robotics, IEEE Transactions on, vol. 24, no. 5, pages 1186 –1198, Oct. 2008.

[Zanchettin 12] A.M. Zanchettin & P. Rocco. *A General User-Oriented Framework for Holonomic Redundancy Resolution in Robotic Manipulators Using Task Augmentation.* Robotics, IEEE Transactions on, vol. 28, no. 2, pages 514 –521, April 2012.

[Zhang 09] Liangjun Zhang, Jia Pan & D. Manocha. *Motion planning of human-like robots using constrained coordination.* In Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on, Dec. 2009.